VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ROZHRANÍ RENDEROVACÍHO STROJE V JAZYCE JAVA

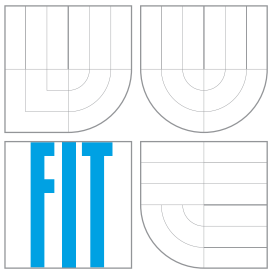
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETER BIELIK

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ROZHRANÍ RENDEROVACÍHO STROJE V JAZYCE JAVA

RENDERING ENGINE JAVA INTERFACE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETER BIELIK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2011

Abstrakt

V Javě existuje nevyplněný prostor, který zde zanechává JEditorPane + HTMLEditorKit, splňující pouze HTML 3.2 a podmnožinu CSS 1. Existují alternativní řešení jako *Lobo / Cobra project*, ale většinou nejeví známky dalšího vývoje. CSSBox je (X)HTML/CSS 2.1 analyzátor, který dokáže vykreslit zpracované data jako jednoduchý bitmapový obrázek. Naším cílem je navrhnout a později implementovat pokročilejší grafické rozhraní pro CSSBox a pokusit se zaplnit prázdný prostor.

Abstract

In Java, there is empty space, which leaves here JEditorPane + HTMLEditorKit, meeting only HTML 3.2 and a subset of CSS 1. There are alternative solutions such as *Lobo / Cobra Project*, but mostly there are no signs of further development. CSSBox is (X)HTML/CSS 2.1 parser that is able to render the processed data as a simple bitmap image. Our goal is to design and later implement advanced graphical interface for CSSBox and try to fill the empty space.

Klíčová slova

java, cssbox, swingbox, vykreslování, rozhraní

Keywords

java, cssbox, swingbox, rendering, interface

Citace

Peter Bielik: Rozhraní renderovacího stroje v jazyce Java, diplomová práce, Brno, FIT VUT v Brně, 2011

Rozhraní renderovacího stroje v jazyce Java

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně s využitím uvedených zdrojú literatúry pod vedením Ing. Radka Burgeta, Ph.D.

.....
Peter Bielik
25.5.2011

Poděkování

Chcel by som sa poďakovať vedúcemu semestrálnej i diplomovej práce,
Ing. R. Burgetovi, Ph.D., za všetky rady a pripomienky, vďaka ktorým mohla moja práca napredovať. Vďaku si zaslúži aj Linus Torvalds, vďaka ktorému vznikol Linux, ako aj James Arthur Gosling, Patrick Naughton a ďalší ako autori jazyka Java.

Obsah

1 Úvod **5**

2 Analýza **6**
2.1 Aplikačné rozhranie jazyka Java . . . 6
2.1.1 Swing . . . 6
2.1.2 Vzťah AWT a Swingu . . . 7
2.1.3 MVC . . . 8
2.1.4 Textový komponent . . . 9
2.1.5 Koncept Document . . . 9
2.1.6 Koncept Editor Kit . . . 10
2.2 Aplikačné rozhranie stroja CSSBox . . . 11
2.2.1 Výsledný model dokumentu . . . 11
2.3 Existujúce riešenia . . . 12
2.3.1 CSSBox . . . 12
2.3.2 JEditorPane + HTMLEditorKit . . . 13
2.3.3 JWebPane . . . 13
2.3.4 JRex . . . 13
2.3.5 Lobo / Cobra project . . . 13
2.3.6 JxBrowser . . . 13
2.3.7 Porovnanie . . . 14
2.4 Špecifikácia požiadaviek . . . 14

3 Návrh **16**
3.1 Predpoklady a závislosti . . . 16
3.2 JEditorPane . . . 16
3.3 Document . . . 17
3.4 ViewFactory . . . 17
3.5 EditorKit . . . 17
3.6 Etapy vývoja . . . 18

4 Implementácia **19**
4.1 Filozofia . . . 19
4.1.1 Pomenovania . . . 19
4.1.2 Hierarchia . . . 20
4.2 SwingBox . . . 21
4.2.1 BrowserPane . . . 21
4.2.2 SwingBoxDocument . . . 21
4.2.3 SwingBoxViewFactory . . . 23

4.2.4 SwingBoxEditorKit . . . 24
4.3 Pomocné objekty . . . 26
4.3.1 Anchor . . . 26
4.3.2 Constants . . . 26
4.3.3 CSSBoxAnalyzer . . . 26
4.3.4 GeneralEvent & GeneralEventListener . . . 27
4.3.5 ContentReader . . . 28
4.3.6 ContentWriter . . . 30
4.3.7 DefaultHyperlinkHandler . . . 30
4.3.8 MouseController . . . 30
4.3.9 ImageLoader . . . 31
4.4 View objekty . . . 32
4.4.1 CSSBoxView . . . 32
4.4.2 RootView . . . 33
4.4.3 ElementBoxView . . . 33
4.4.4 BlockBoxView . . . 34
4.4.5 InlineBoxView . . . 34
4.4.6 ViewportView . . . 34
4.4.7 DelegateView . . . 35
4.4.8 TextBoxView . . . 35
4.4.9 ReplacedContent & ReplacedImage . . . 36
4.4.10 InlineReplacedBoxView & BlockReplacedBoxView . . . 37
4.4.11 Ostatné View Objekty . . . 37
4.5 Demo aplikácia . . . 37

5 Testovanie **44**
5.1 Známe chyby a nedostatky . . . 44

6 Záver **46**

A Obsah CD **48**

B Ukážka použitia **49**

Seznam obrázků

2.1 Znázornenie Java Foundation Classes . . . 7
2.2 Hierarchia triedy JComponent. . . . 8
2.3 Vzor MVC. . . . 8
2.4 Vzor MVC v podaní Swingu. . . . 9
2.5 Hierarchia dokumentov. . . . 9
2.6 Hierarchia dedenia EditorKitu. . . . 11
2.7 Hierarchia dedenia triedy Box. . . . 12

3.1 Konceptuálny návrh. . . . 18

4.1 Hierarchia balíčkov tried. . . . 20
4.2 Diagram triedy BrowserPane. . . . 22
4.3 Diagram triedy SwingBoxDocument a DelegateElement. . . . 23
4.4 Diagram triedy SwingBoxViewFactory. . . . 23
4.5 Diagram triedy SwingBoxEditorKit. . . . 25
4.6 Diagram triedy Anchor. . . . 26
4.7 Diagram rozhrania CSSBoxAnalyzer. . . . 27
4.8 Diagram triedy DefaultAnalyzer. . . . 27
4.9 Diagram triedy GeneralEvent a GeneralEventListener. . . . 28
4.10 Diagram triedy ContentReader. . . . 29
4.11 Diagram triedy ContentWriter. . . . 30
4.12 Diagram triedy DefaultHyperlinkHandler. . . . 31
4.13 Diagram triedy MouseController. . . . 31
4.14 Diagram triedy ImageLoader. . . . 32
4.15 Diagram rozhrania CSSBoxView. . . . 33
4.16 Diagram triedy ElementBoxView. . . . 38
4.17 Diagram triedy BlockBoxView. . . . 39
4.18 Diagram triedy InlineBoxView. . . . 39
4.19 Diagram triedy ViewportView. . . . 39
4.20 Diagram triedy DelegateView. . . . 40
4.21 Diagram triedy TextBoxView. . . . 41
4.22 Diagram triedy ReplacedContent a ReplacedImage. . . . 42
4.23 Diagram triedy BlockReplacedBoxView. . . . 43

5.1 Ukážka testovacej aplikácie s výsledkom v BrowserPane. . . . 44

Seznam tabulek

2.1 Porovnanie existujúcich riešení 14

3.1 Prehlad dôležitých metód EditorKitu. 17

Kapitola 1

Úvod

CSSBox je open-source projekt pre spracovanie (X)HTML/CSS, ktorého primárnym cieľom je poskytnutie informácií o rozložení a obsahu spracovanej stránky. Umožňuje i vizualizáciu spracovaných dát, no jedná sa len o vygenerovaný obrázok s ktorým sa ďalej už nedá rozumne pracovať (napr. nemožnosť kopírovať text).
Táto práca si dáva za cieľ navrhnúť a implementovať pokročilejšie grafické rozhranie pre zobrazenie spracovaných dát, pričom bude kladený dôraz na správne zobrazenie obsahu.

Kapitola 2

Analýza

Táto kapitola pojednáva o aplikačnom rozhraní jazyka Java a programu CSSBox, poskytuje prehľad o existujúcich riešeniach a na záver je formulovaná špecifikácia požiadaviek.

2.1 Aplikačné rozhranie jazyka Java

Korene Javy možno vystopovať až k projektu *Oak*[1], ktorý začal v roku 1991. Pôvodne mala Java obohacovať televízne prijímače, rekordéry a rôzne multimediálne zariadenia o interaktívne funkcie. Neskôr sa ukázalo, že táto oblasť nie je dostatočne rentabilná a tak sa zmenila orientácia na Internet. Spoločne s virtuálnym strojom, ktorý pôsobí ako abstrakčná vrstva medzi dodaným kódom a platformou, bolo docela jednoduché stiahnuť a spustiť applet vo svojom prehliadači. Navyše, virtuálny stroj spravuje prístup k systému a jeho prostriedkom, čo predstavuje istú formu bezpečnosti. Po veľkom úspechu appletov, bola Java postupne rozšírená o funkcionalitu a možnosti známe z programov pre stolné počítače či servery. Ako pribúdala funkčnosť, pribúdali aj požiadavky, najmä na grafické užívateľské rozhranie. A tak popri AWT začal vznikať Swing. Dnes, Java patrí k jedným z najrozšírenejších jazykov.

2.1.1 Swing

Swing [6] možno označiť ako sadu prispôsobiteľných grafických prvkov, ktorých vzhľad (Look-and-Feel[2]) možno za behu meniť, užívateľské rozhranie novej generácie. Označenie Swing nie je akronym ako by sa zdalo, ale reprezentuje voľbu vývojárov na počiatku projektu v roku 1996. Swing je v skutočnosti súčasťou omnoho širšej rodiny Java produktov, známych ako Java Foundation Classes (JFC), ktoré zahŕňajú mnohé z vlastností Internet Foundation Classes (IFC) od spoločnosti Netscape ako aj aspekty návrhu z divízie IBM Taligent a Lighthouse Design. API Swingu dosiahlo beta štádium v neskoršej polovici roku 1997 a prvý raz bolo zverejnené v marci 1998. V tom čase, Swing 1.0, obsahoval takmer 250 tried a 80 rozhraní.

Java Foundation Classes [6] je sadou knižníc, ktorá okrem Swingu pozostáva z nasledovného:

[1] Veľmi príjemné čítanie: http://web.archive.org/web/20010610045734/http://java.sun.com/nav/whatis/storyofjava.html

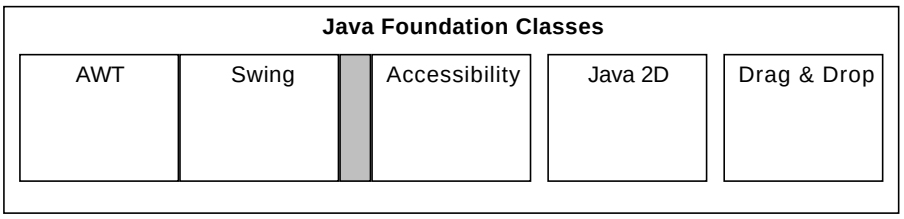
[2] Skrátene označované ako L&F/LaF. Look označuje vzhľad komponentu (presnejšie, inštancie triedy JComponent), Feel označuje spôsob, akým sa správajú komponenty, http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html

AWT - Abstract Window Toolkit, predstavuje základnú sadu nástrojov pre tvorbu GUI, ktoré sú prítomné v každej verzii JDK. Na rozdiel od Swingu, AWT používa ťažké (heavyweight) komponenty.

Accessibility - Knižnica venovaná dostupnosti, poskytuje asistenciu tým užívateľom, ktorí majú problémy s interakciou užívateľského rozhrania a využívajú špecializované zariadenia ako braillova klávesnica.

2D API - Súbor tried pre vykresľovanie komplexných tvarov, farieb, štýlov a fontov. Nie je súčasťou Swingu.

Drag & Drop - API umožňujúce implementovať drag & drop elementy na prenos informácie medzi Java aplikáciou a natívnym prostredím.



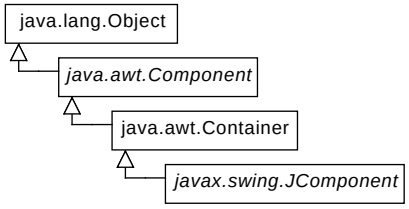
Obrázek 2.1: Znázornenie Java Foundation Classes

2.1.2 Vzťah AWT a Swingu

Skutočnosť je taká, že Swing [6] je postavený na jadre knižnice AWT a neobsahuje žiadne platformovo závislé inštrukcie, vďaka čomu môže byť aplikácia jednoducho distribuovaná na rozličné podporované platformy už od verzie Javy 1.1.5.
Ako je spomenuté v [3], AWT využíva heavyweight komponenty. To znamená, že sa jedná o taký komponent, ktorý je vlastníkom istého množstva natívnych zobrazovacích prostriedkov (označovaných ako *peer*).
Lightweight komponent si len *„požičiava“* natívne prostriedky od svojho predchodcu a žiadne nevlastní. Práve takéto komponenty využíva Swing, čo má za následok:

- Lepšie využitie prostriedkov.
- Lepšia konzistencia na podporovaných platformách: Swing je písaný len v Jave.
- Lepšia podpora pre Look-and-Feel.

S výnimkou prvkov najvyššej úrovne, kontajnerov (napr. JFrame, JDialog,...), sú vo Swingu [9] všetky komponenty, ktorých mená začínajú na „**J**“, odvodené od triedy `JComponent` (napr. JButton, JPanel, JLabel,...). Tento prvok je odvodený od tried `Container` a `Component`, ktoré už spadajú pod AWT.



Obrázek 2.2: Hierarchia triedy JComponent.

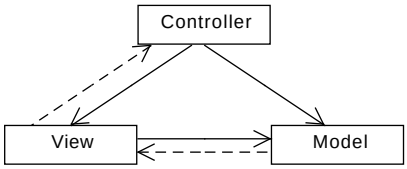
2.1.3 MVC

Vnútorná [8] architektúra swingu vychádza hlavne z MVC[3], ktorý možno datovať od čias SmallTalku[4]. Hlavnou myšlienkou tohto vzoru je oddelenie logiky pre riadenie užívateľského rozhrania od biznis logiky. Za týmto účelom je komponenta rozdelená na tri separátne časti:

model - reprezentuje dáta pre aplikáciu.

view - vizuálna reprezentácia dát z modelu.

controller - preberá užívateľské vstupy od view a transformuje ich na akcie, ktoré manipulujú s modelom.

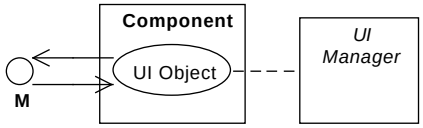


Obrázek 2.3: Vzor MVC.
Plné šípky predstavujú priamu asociáciu, prerušované nepriamu asociáciu (napr. vzor Observer).

V prvotnej verzii Swingu [4], boli komponenty rozdelené podľa filozofie MVC, teda každý komponent mal separátny objekt predstavujúci model a delegoval svoju L&F implementáciu do separátnych View a Controller objektov. Čoskoro sa ukázalo, že toto rozdelenie v praxi nepracuje dobre. Objekty implementujúce časť view a controller podľa vzoru MVC, vyžadujú úzku previazanosť (je veľmi obtiažné napísať všeobecný controller, ktorý nevie takmer nič o objekte, ktorý implementuje časť view). Preto boli tieto dve entity zlúčené do jedného objektu.

Ako vidno, architektúra Swingu sa voľne inšpiruje vzorom MVC, no nedodržiava ho striktne... Vo svete Swingu je tento nový, kvázi MVC model, niekedy označovaný ako aj model oddeliteľnej architektúry (separable model architecture). Tento nový prístup oddeľuje modelovú časť komponentu rovnako, ako vzor MVC, no časti view a controller každého komponentu spojuje do jedného UI (User-Interface) objektu.

[3] Ďalšie zaujímavé čítanie nielen o MVC: http://www.martinfowler.com/eaaDev/uiArchs.html
[4] http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html



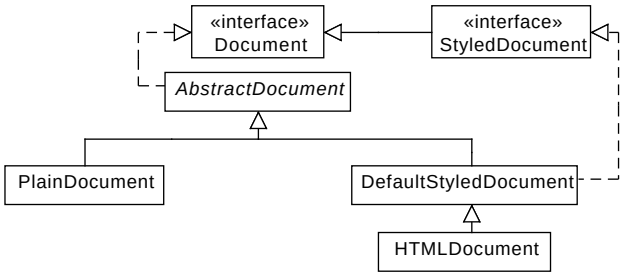
Obrázek 2.4: Vzor MVC v podaní Swingu.

2.1.4 Textový komponent

Trieda `JTextComponent`[5] [10] predstavuje základ pre všetky textovo orientované komponenty vo Swingu. Vo svojom kóde sa snaží byť kompatibilná s `java.awt.TextComponent`. Táto trieda poskytuje nasledovné vlastnosti pre všetkých svojich potomkov, ktoré môžu byť dodatočne upravované:

- model - v tomto prípade známy ako *dokument*, predstavuje obsah komponentu.
- view - vykresľuje obsah komponentu na obrazovku.
- controller - v tomto prípade *editačná sada – editor kit*, číta a zapisuje obsah, implementuje editačné funkcie a akcie.
- podporu pre *nekonečné* undo a redo funkcie.
- podporu pre objekty typu Caret, CaretChangeListener,...

2.1.5 Koncept Document



Obrázek 2.5: Hierarchia dokumentov.

Dokument - inštancia triedy [10], ktorá implementuje rozhranie `Document`[6] je modelom textovej komponenty a poskytuje nasledovné služby:

- Uchováva text. Dokument ukladá svoj textový obsah vo forme objektov typu Element[7], vďaka ktorým dokáže reprezentovať ľubovolnú logickú štruktúru textu, ako sú odstavce či bloky textu zdieľajúce jednotný štýl formátovania.

[5] http://download.oracle.com/javase/6/docs/api/javax/swing/text/JTextComponent.html
[6] http://download.oracle.com/javase/6/docs/api/javax/swing/text/Document.html
[7] http://download.oracle.com/javase/6/docs/api/javax/swing/text/Element.html

- Umožňuje editáciu textu prostredníctvom metód `remove` a `insertString`.
- Objekty typu `DocumentListener` [8] a `UndoableEditListener`[9] sú informované o zmenách v texte.
- Spravuje objekty typu `Position`[10], ktoré uchovávajú konkrétne pozície v rámci textu. Hlavnou myšlienkou je skryť implementačné detaily dokumentu, ale zároveň umožniť špecifikovanie pozície v rámci daného dokumentu. Objekty sú spravované aj počas úpravy textu.
- Umožňuje získanie podrobnejších informácií o texte, ako je dĺžka či segment daného reťazca.

Na obrázku 2.5 je znázornená hierarchia dokumentov z balíčka `javax.swing.text`. Rozhranie [10] `StyledDocument`[11] rozširuje `Document` o schopnosť dekorovať text rôznymi štýlmi a vytvárať tak bohatší obsah. `PlainDocument` je štandardným dokumentom pre všeobecné textové polia. Poskytuje základný kontajner pre text s jednotným fontom pre celý obsah bez možnosti aplikovať pokročilejšie štýly. Na druhej strane, `DefaultStyledDocument` je štandardným dokumentom pre `JTextPane`. V tomto, ale aj v prípade prvku `JEditorPane`, inštancia konkrétneho dokumentu závisí aj na type obsahu. Ak pre načítanie obsahu použijeme metódu `setPage`, môže dôjsť ku zmene inštancie používaného dokumentu.

2.1.6 Koncept Editor Kit

Abstraktná trieda `EditorKit`[12] [10] predstavuje základnú triedu zastrešujúcu vytváranie dokumentu, spravovanie akcií, ako aj vizuálnu reprezentáciu obsahu dokumentu. Navyše, EditorKit vie, ako má správne čítať, alebo zapisovať dáta reprezentujúce obsah dokumentu v danom prúde dát. Každý typ dokumentu vyžaduje svoju vlastnú implementáciu triedy EditorKit.
Zobrazenie obsahu dokumentu je vykonané prostredníctvom triedy EditorKit s pomocou od `ViewFactory`[13]. Pre každý objekt typu Element patriaci do dokumentu, ViewFactory rozhodne, ktorý objekt typu View bude vytvorený a zobrazený. Pre každý objekt typu Element, existuje korešpondujúci objekt typu View. Sú však prípady, kedy skupina elementov tvorí jeden ucelený objekt View, napr. tabulka.

Swing [10] poskytuje nasledovné EditorKity:

DefaultEditorKit - spracováva obyčajný text a poskytuje základnú funkčnú výbavu.

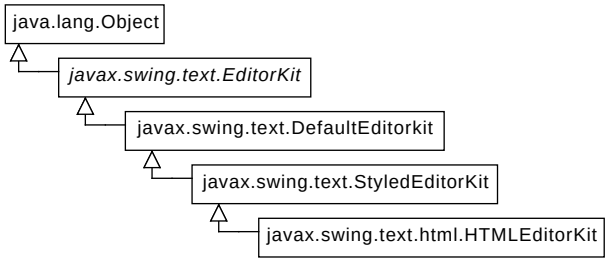
StyledEditorKit - spracováva text obohatený o štýly a poskytuje funkčnú výbavu pre prácu s takýmto textom. Je to východzí EditorKit pre JTextPane.

HTMLEditorKit - tento EditorKit je určený výhradne pre prácu s HTML, no má isté obmedzenia.

RTFEditorKit - určený pre prácu s formátom RTF.

[8] http://download.oracle.com/javase/6/docs/api/javax/swing/event/DocumentListener.html
[9] http://download.oracle.com/javase/6/docs/api/javax/swing/event/UndoableEditListener.html
[10] http://download.oracle.com/javase/6/docs/api/javax/swing/text/Position.html
[11] http://download.oracle.com/javase/6/docs/api/javax/swing/text/StyledDocument.html
[12] http://download.oracle.com/javase/6/docs/api/javax/swing/text/EditorKit.html
[13] http://download.oracle.com/javase/6/docs/api/javax/swing/text/ViewFactory.html

Na obrázku 2.6 je znázornená hierarchia medzi jednotlivými EditorKitmi. Každý z týchto EditorKitov je registrovaný s triedou JEditorPane a asociovaný so správnym typom obsahu, ktorý dokáže spracovávať. JEditorPane potom vyberie správny EditorKit, vďaka ktorému sa efektívne transformuje na editor príslušného obsahu súboru. Vďaka tomuto sa z komponentu JEditorPane stáva „univerzálny vojak“...



Obrázek 2.6: Hierarchia dedenia EditorKitu.

2.2 Aplikačné rozhranie stroja CSSBox

CSSBox [2] je (X)HTML/CSS renderovací stroj napísaný výhradne v jazyku Java. Hlavným cieľom je poskytnutie kompletnej informácie (a umožniť ďalšie spracovanie) o obsahu aktuálne spracovávanej stránky a jej rozložení. CSSBox momentálne podporuje aj zobrazenie spracovaného obsahu.

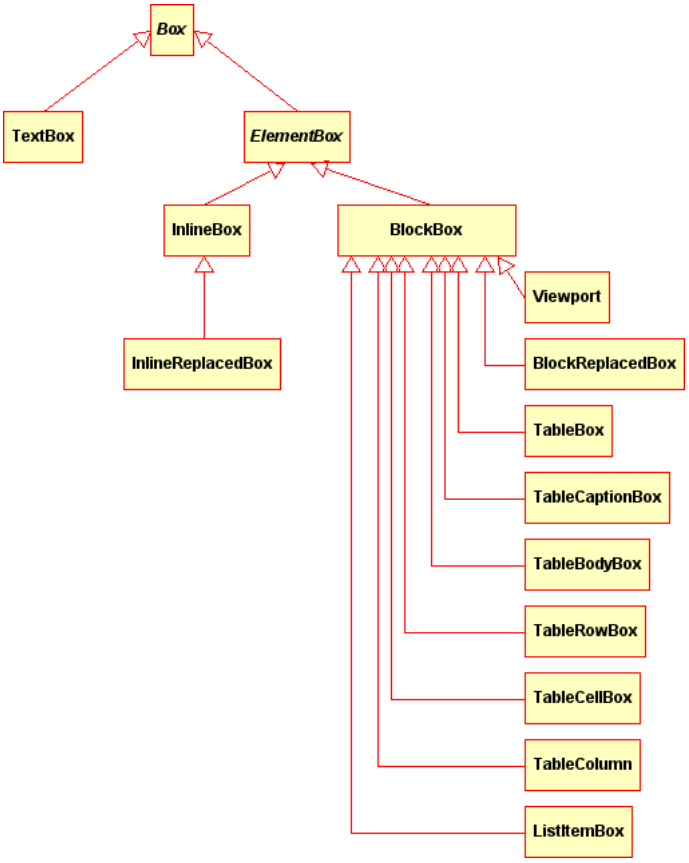
Vstupom pre renderovací stroj je strom objektovej reprezentácie modelu dokumentu (DOM - Document Object Model). Následne sú načítané všetky štýly, na ktoré sa dokument odkazuje a vypočíta sa efektívny štýl pre každý element. Výstupom je objektovo-orientovaný model rozloženia stránky. Tento model možno použiť na zobrazenie stránky, no najmä na ďalšie spracovanie algoritmami ako je analýza rozloženia či extrakcia informácie. Pre zobrazenie je možné využiť objekt `BrowserCanvas`[14], ktorý je odvodený od triedy `javax.swing.JPanel`. Preto je možné ho jednoducho použiť ako Swing komponentu pri vytváraní užívateľského rozhrania. Velkosť komponentu je automaticky prispôsobená výsledku rozloženia dokumentu.

2.2.1 Výsledný model dokumentu

Výsledné rozloženie dokumentu [2] je reprezentované ako strom objektov typu `Box`[15]. Každý takýto objekt vytvára obdĺžnikovú oblasť vo výslednej stránke, čo zodpovedá konkrétnemu HTML elementu. Môžu sa vyskytnúť aj viaceré boxy, korešpondujúce práve k jednému elementu. Napríklad odstavec `<p>` s viacerými riadkami, je rozdelený na boxy predstavujúce jednotlivé riadky. Box môže tiež pozostávať z potomkov daného boxu, alebo môže zodpovedať konkrétnej časti obsahu dokumentu, teda časti textu, alebo obrázku.

Každý box predstavuje objekt, ktorý dedí od abstraktnej triedy `Box`. Existuje niekoľko typov boxov, ktoré zodpovedajú hodnote CSS vlastnosti display pre daný element. Napríklad `p {display:inline}` spôsobí, že element vygeneruje `InlineBox`, teda žiadne zalomenie riadka ani pred, ani za elementom. Koreň stromu je vždy reprezentovaný ako objekt

[14] http://cssbox.sourceforge.net/api/org/fit/cssbox/layout/BrowserCanvas.html
[15] http://cssbox.sourceforge.net/api/org/fit/cssbox/layout/Box.html



Obrázek 2.7: Hierarchia dedenia triedy Box.

`Viewport`[16] a predstavuje výrez obsahu dokumentu v prehliadači. Viewport má vždy jediného potomka, *root box*, ktorý predstavuje koreňový HTML element dodaný na vykresľovanie objektu `BrowserCanvas`. Zvyčajne sa jedná o HTML element `<body>`.

2.3 Existujúce riešenia

Táto časť sa zaoberá stručnou analýzou existujúcich riešení a ich hodnotením. Treba však zdôrazniť, že sa pohybujeme medzi (X)HTML/CSS analyzátormi a renderovacími strojmi, hraničiacimi s internetovými prehliadačmi.

2.3.1 CSSBox

Program, ktorý predstavuje základ tejto práce, je napísaný v čistej Jave v duchu open-source. Podporuje spracovanie (X)HTML a CSS 2.1, na analyzovanie ktorých využíva knižnicu `jStyleParser`[17]. Podporované je aj zobrazenie spracovaného obsahu (komponent BrowserCanvas), no výsledkom je len bitmapový obrázok, s ktorým sa už nedá ďalej pracovať. Posledná verzia, 3.2, je dostupná z dňa 8.2.2011. K správnej funkčnosti je potrebná Java 1.6. Vývoj naďalej pokračuje...
URL: http://cssbox.sourceforge.net

[16] http://cssbox.sourceforge.net/api/org/fit/cssbox/layout/Viewport.html
[17] http://cssbox.sourceforge.net/jstyleparser

2.3.2 JEditorPane + HTMLEditorKit

Jedná sa o štandardný ovládací prvok z balíčka `javax.swing`, dostupný v každom riadnom JRE/JDK. Trieda `javax.swing.text.html.HTMLEditorKit` sa stará o spracovanie a analýzu kódu. Výsledkom je dobre vyzerajúci výstup. Treba zdôrazniť, že podpora HTML je limitovaná len do verzie 3.2 a štýly len na podmnožinu CSS 1[18], čo v čase kraľovania verzie 4.01 a chystaného nástupu verzie 5, je nepostačujúce. V mnohých prípadoch nie sú podporované relatívne jednotky ako napr. `p { margin-top: 10% }`. Posledná verzia, 1.142, je dostupná z dňa 23.3.2010, respektíve, 1.137 z rovnakého dňa.

2.3.3 JWebPane

Je predstaviteľom budúceho, štandardného prvku pre Swing, ktorý by mal spĺňať najnovšie štandardy vrátane Javascriptu. Jadrom komponentu by mal byť WebKit, integrovaný priamo do JDK/JRE. Po oficiálnom predstavení komponentu na konferencií JavaOne[19] v r. 2008 nastalo ticho a budúcnosť je nejasná...
URL: http://weblogs.java.net/blog/2008/05/29/introducing-jwebpane-component

2.3.4 JRex

Komponent vyvinutý pod hlavičkou spoločnosti Mozilla, predstavuje zlepenec Javy a natívneho kódu. Presnejšie, k užívateľskému rozhraniu v Jave pribaľuje funkcionalitu natívnej knižnice GECKO 1.4+ . K spusteniu je potrebný Windows, alebo Linux a JDK 1.4.2+ . Posledná verzia pochádza z 27.4.2005 a nie je známe ďalšie pokračovanie projektu.
URL: http://jrex.mozdev.org
URL: http://jrex.mozdev.org/docs.html

2.3.5 Lobo / Cobra project

Cobra - predstavuje Java implementáciu stroja pre analyzovanie, spracovanie a vykresľovanie HTML 4, CSS 2 a Javascriptu. Lobo je jednoduchou Java aplikáciou v roli internetového prehliadača postaveného na Cobre. Pre správne spustenie je potrebné JRE 1.5+, pre podporu JavaFX, JRE 1.6u10+. Posledná verzia, 0.98.4, je datovaná 19.1.2009. Aj napriek tomu, že na stránke http://sourceforge.net/projects/xamj/ je uvedené označenie beta, jedná sa o veľmi pokročilý program. Okolo projektu je mimoriadne ticho a pravdepodobne sa autor vzdal vývoja.
URL: http://lobobrowser.org/cobra.jsp
URL: http://lobobrowser.org/java-browser.jsp

2.3.6 JxBrowser

Je komerčný produkt podobný projektu JRex, ktorý kombinuje Javu a natívny kód. Pre spracovanie dát dokáže využívať GECKO, WebKit a Trident. Pre spustenie vyžaduje Windows, Macintosh OS X 10.5+, alebo Linux a JDK 1.6u12+. Najnovšia verzia, 2.6, je datovaná 17.10.2010. Nakoľko sa jedná o komerčný produkt, cena jednej licencie je 980,- eur (k 26.12.2010). V prípade tohto projektu, vývoj jednoznačne napreduje...

[18] http://download.oracle.com/javase/6/docs/api/javax/swing/text/html/CSS.html
[19] Podrobnosti od strany 27, http://developers.sun.com/learning/javaoneonline/2008/pdf/TS-6610.pdf

URL: http://www.teamdev.com/jxbrowser/

2.3.7 Porovnanie

	HTML-EditorKit	CSSBox	JWebPane	JRex	Cobra	JxBrowser
Podporuje	HTML 3.2, CSS 1	(X)HTML, CSS 2.1	±	±	HTML 4, CSS 2	±
Platforma	∗	∗	∗	†	∗	‡
Open-source	+	+	+	+	+	-
Cena	0	0	0	0	0	980 eur
Implementácia	Java	Java	Java	Java + native	Java	Java + native

Tabulka 2.1: Porovnanie existujúcich riešení

± - Záleží na schopnostiach použitej knižnice.
∗ - Solaris, MS Windows, Linux, Macintosh OS X
‡ - MS Windows, Linux, Macintosh OS X
† - MS Windows, Linux

V Jave existuje nevyplnený priestor, ktorý tu zanecháva JEditorPane + HTMLEditorKit, spĺňajúci len HTML 3.2 a podmnožinu CSS 1. Sľubovaný JWebPane, ako náhrada vyššie uvedených objektov, sľubujúci bohatý webový obsah, zatiaľ existuje len na papieri. . . Doposiaľ najlepším riešením, ktoré by bolo dostupné v čistej Jave a duchu open-source, je *Lobo / Cobra project*, no tento projekt nejaví známky ďalšieho vývoja aj napriek tomu, že na stránke projektu je hlásených množstvo chýb od svojich užívateľov, ktoré by bolo vhodné opraviť.

2.4 Špecifikácia požiadaviek

Na vytvárané programové riešenie sú kladené nasledovné požiadavky:

- Bude vytvorený komponent rozhrania Swing, obdĺžnikového tvaru, v ktorom sa bude zobrazovať spracovaný obsah. Užívateľ nebude môcť priamo upravovať obsah dokumentu.
- Komponent nebude obsahovať žiadne menu, tlačítka, nástrojové lišty či iné ovládacie prvky známe z internetových prehliadačov.
- Zobrazený obsah nebude reprezentovaný *„jednoliatym bitmapovým obrázkom“*. Komponent sa pokúsi správne zobraziť daný typ obsahu, teda textový obsah bude skutočným textom, obrázok obrázkom a podobne. V prípade nepodporovaného obsahu, bude zobrazené vhodné upozornenie.
- Zobrazený text bude možné kopírovať do systémovej schránky.

- CSSBox bude použitý pre analyzovanie a spracovanie rozloženia obsahu dokumentu. Komponent spracuje tie elementy daného dokumentu, ktoré CSSBox dokáže reprezentovať na svojom výstupe. Predovšetkým sa jedná o štrukturovaný a dekorovaný text, obrázky, tabuľky, zoznamy a odkazy.
- V prípade aktivovania odkazu, bude spracovaný odkazovaný dokument. Obsah bude vždy zobrazený v aktuálnom okne, ktoré predstavuje viditeľná plocha komponentu. Nové okná nebudú vytvárané.
- Výsledné riešenie bude vo forme knižnice.

Doplnkové vlastnosti, ktoré vo výslednom produkte nemusia byť úplne, prípadne vôbec implementované:

- Podpora Javascriptu prostredníctvom knižnice Rhino[20].
- Rozhranie pre zasielanie stavu spracovania aktuálneho dokumentu. Signalizuje, začiatok, ukončenie a chybu počas spracovania dokumentu.

[20]Posledná verzia, 1.7R2, z 22.3.2009: http://www.mozilla.org/rhino

Kapitola 3

Návrh

Ako zo špecifikácie požiadaviek vyplýva, výsledný komponent musí byť použiteľný vo Swingu, teda musí vychádzať z triedy JComponent. Najvhodnejším kandidátom je *JEditorPane*, ktorý už v základe vyhovuje a nebude potreba veľkých zmien. Toto rozhodnutie nám prináša už hotovú architektúru a nemusíme nič nové vymýšľať. Musíme však vytvoriť požadované triedy, pričom jadrom práce bude EditorKit, spracovanie obsahu a vykresľovanie pomocou View objektov (vzhľadom na veľkosť práce, je vhodné myslieť na znovu použiteľnosť kódu). Požiadavkou je aj výstup práce vo forme knižnice. Toto bude zabezpečené formou samostatného Jar súboru, z ktorého budú triedy programovo dostupné. CSSBox však nebude obsiahnutý, nakoľko sa vyvíja samostatne. Vďaka tomu bude umožnená jednoduchá aktualizácia knižníc v prípade novej verzie. Pre lepšiu identifikáciu, projekt ponesie pracovný názov *SwingBox*.

3.1 Predpoklady a závislosti

- Java - primárnym cieľom je J2SE od spoločnosti Oracle (Sun), JRE/JDK 1.6. Vývoj bude prebiehať na JDK 1.6.0u23. Sekundárnym cieľom je kompatibilita s OpenJDK 6.
- Operačný systém - systémy, ktoré sú kompatibilné s Oracle (Sun) JRE/JDK 1.6. Vo všeobecnosti sú to systémy Solaris, Windows a Linux. Vývoj bude prebiehať na systéme Linux, Fedora 13.
- Knižnice - CSSBox 3.2 spolu s vlastnými knižnicami.
- Užívateľ - budeme rozumieť Java programátora, ktorý má dostatok skúseností s používaním komponentov Swingu.

3.2 JEditorPane

Splňuje prvý a posledný bod v špecifikácií požiadaviek, teda poskytuje možnosti pre zobrazenie obsahu, dokáže zabrániť nechcenej editácií obsahu a podporuje prácu s odkazmi. Toto všetko je už dostupné v štandardnej implementácií. Je možné, že v konečnej implementácií pribudne i ďalšia funkcionalita nad rámec požiadaviek, ako napr. zobrazovanie popiskov.

3.3 Document

Je modelom tohto komponentu z pohľadu MVC. Azda jedinou a najdôležitejšou požiadavkou je, že musí byť odvodený od `DefaultStyledDocument`, pre správnu funkčnosť. Východzia implementácia už poskytuje všetky potrebné metódy, no niektoré budú delegované kvôli viditeľnosti (napr. `void create(ElementSpec[] data)`). Nie je však vylúčené, že počas implementácie sa vyskytnú ďalšie úpravy.

3.4 ViewFactory

ViewFactory je rozhranie s jedinou metódou, `public View create(Element e)`. Pointou je vytvorenie správneho objektu typu View pre každý Element z dokumentu (napr. `LabelView(Element e)` pre zobrazenie textu). Tieto objekty [7] predstavujú *pohľad* na model, teda sú zodpovedné za *výzor* textového komponentu. Zo špecifikácie požiadaviek, budú napĺňať tretí bod. Neexistuje žiadna univerzálna implementácia, ktorá by sa dala využiť. Trieda musí byť vytvorená od základu.

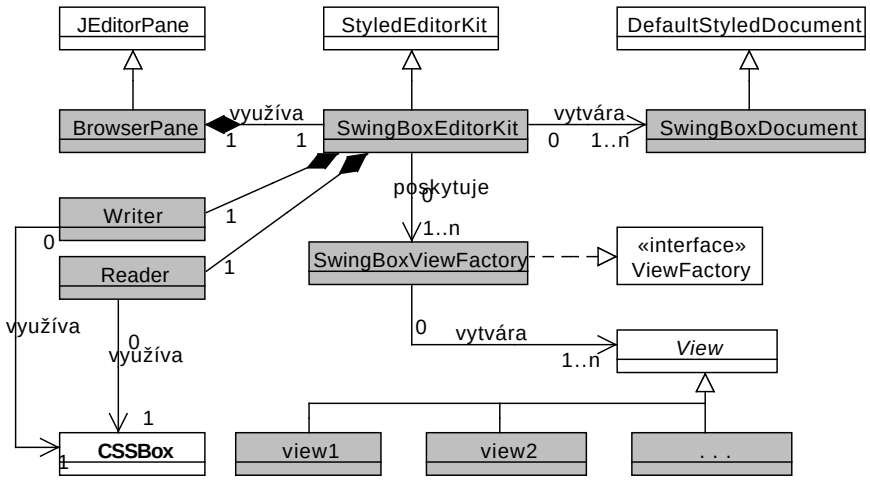
3.5 EditorKit

Pre správnu funkčnosť je potrebné:

- Implementovať EditorKit odvodený od triedy StyledEditorKit (pretože budeme používať JEditorPane).
- Definovať a vytvoriť dokument.
- Vytvoriť ViewFactory.
- Implementovať vlastné View objekty.
- Vytvoriť „reader“, respektíve „writer“ pre čítanie a zapisovanie obsahu. V tomto projekte má implementácia triedy „reader“ väčšiu prioritu než „writer“.
- JEditorPane musí vedieť o novom EditorKite.

Typ	**Metóda**
Document	createDefaultDocument()
void	install(JEditorPane c)
void	deinstall(JEditorPane c)
Caret	createCaret()
String	getContentType()
ViewFactory	getViewFactory()
void	read(Reader in, Document doc, int pos)
void	write(Writer out, Document doc, int pos, int len)
Object	clone()
Action[]	getActions()

Tabulka 3.1: Prehlad dôležitých metód EditorKitu.



Obrázek 3.1: Konceptuálny návrh.

Metóda `createDefaultDocument()`, bude vždy vracať novú inštanciu používaného dokumentu. Výsledkom volania metódy `getViewFactory()` bude inštancia triedy, ktorá implementuje rozhranie ViewFactory. Metóda `getContentType()` určuje, aký obsah editor kit dokáže spracovať. V našom prípade, metóda bude vracať hodnotu `text/html`. Kedže SwingBox nebude podporovať vytváranie či úpravu obsahu, nie je potreba poskytovať kurzor (caret). Editor kit nebude vyžadovať žiadne špeciálne zaobchádzanie počas svojej aktivácie či deaktivácie a preto metódy install(JEditorPane c) a deinstall(JEditorPane c) môžu ostať v pôvodnej implementácií. Bez zmien môžu ostať aj clone() a getActions(). Je takmer isté, že metóda pre načítanie obsahu, read(. . .), bude značne komplikovaná. Najvhodnejšie bude presunúť celú logiku spracovania do samostatnej, jednoúčelovej triedy, ktorá bude využívať CSSBox, ako je uvedené v piatom bode špecifikácie požiadavkov. Podobne to bude aj s metódou write. Na obrázku 3.1 je znázornený konceptuálny návrh, pričom tmavou farbou sú zvýraznené triedy, ktoré SwingBox bude implementovať.

3.6 Etapy vývoja

Vývoj bude prebiehať v nasledovných etapách. Každá etapa sa zameriava na implementáciu danej problematiky.

- m1 - dekorácia textu (farba, veľkosť, písmo, apod.)
- m2 - štrukturovanie a odstavce
- m3 - obrázky
- m4 - odkazy
- m5 - tabuľky a zoznamy

Kapitola 4

Implementácia

V tejto kapitole je opísaná implementácia SwingBoxu spolu s problémami a postupom riešenia.

4.1 Filozofia

Jedným z cieľov bol dôraz na vysokú súdržnosť a nízku previazanosť, ako aj maximalizovanie znovu použiteľnosti kódu. CSSBox už obsahuje prvky, ktoré v danom grafickom kontexte dokážu vykresliť svoj obsah, čo SwingBox využíva v maximálnej miere. Tieto prvky (presnejšie triedy v balíčku `org.fit.cssbox.layout`) majú svoju hierarchiu, čo sa odzrkadlilo aj na hierarchií tried View objektov (balíček `org.fit.cssbox.swingbox.view`), ktoré ju priamo kopírujú až na triedu Box. Postupom času sa ukázalo, že problémom sú pozície. View objekty pracujú s relatívnymi pozíciami, ktoré sa prepočítavajú na absolútne, zatiaľ čo CSSBox vo výslednom spracovaní vytvorí objekty s absolútnymi pozíciami. Robiť spätný prepočet na relatívne pozície je neefektívne, navyše absolútne pozície je možné okamžite využiť pri vykresľovaní a nezdržovať sa prepočtami. Rozhodnutie používať absolútne pozície však prináša nutnosť nanovo implementovať `BoxView`[1] - štandardný objekt slúžiaci ako kontajner pre View objekty, ktorý je odvodený od abstraktnej triedy `CompositeView`[2]. Vzhľadom na hierarchiu Boxov je najvhodnejšie, aby CompositeView implementoval ElementBoxView, od ktorého dedia všetky objekty, okrem TextBoxView. Pri toľkom množstve objektov je vhodné, aby sme mali spôsob, ako ich dať do kolekcie všetky, prípadne, aby sme boli schopný rozlišovať „naše“ objekty od ostatných pomocou `instanceof`. Z tohto dôvodu každý View objekt patriaci do SwingBoxu, implementuje rozhranie `CSSBoxView`, ktoré podobne, ako rozhranie `Serializable`[3] slúži najmä na identifikáciu objektov (v tomto zmysle nahradzuje abstraktnú triedu Box v CSSBoxe, ktorá je umiestnená najvyššie v hierarchií dedenia).

4.1.1 Pomenovania

Ako už bolo spomenuté, bude potrebné definovať Document, EditorKit a ViewFactory. To všetko sú veľmi všeobecne pomenovania. Keďže pracujeme s (X)HTML, je úplne prirodzené dať pomenovania HTMLDocument, HTMLEditorKit či HTMLViewFactory, no tieto už existujú v rámci distribúcie JRE/JDK. Pre lepšiu prehľadnosť a jednoznačnosť boli zvolené

[1] http://download.oracle.com/javase/6/docs/api/javax/swing/text/BoxView.html
[2] http://download.oracle.com/javase/6/docs/api/javax/swing/text/CompositeView.html
[3] http://download.oracle.com/javase/6/docs/api/java/io/Serializable.html

mená *SwingBoxDocument*, *SwingBoxEditorKit*, *SwingBoxViewFactory*. Hlavný komponent nesie označenie *BrowserPane* po vzore JEditorPane, z ktorého je odvodený. V podobnom duchu sú pomenované aj View objekty, ktoré už v názve reflektujú, aký príslušný Box reprezentujú: *ElementBoxView*, *BlockBoxView*, *TextBoxView*, *ViewportView*, atd.

4.1.2 Hierarchia

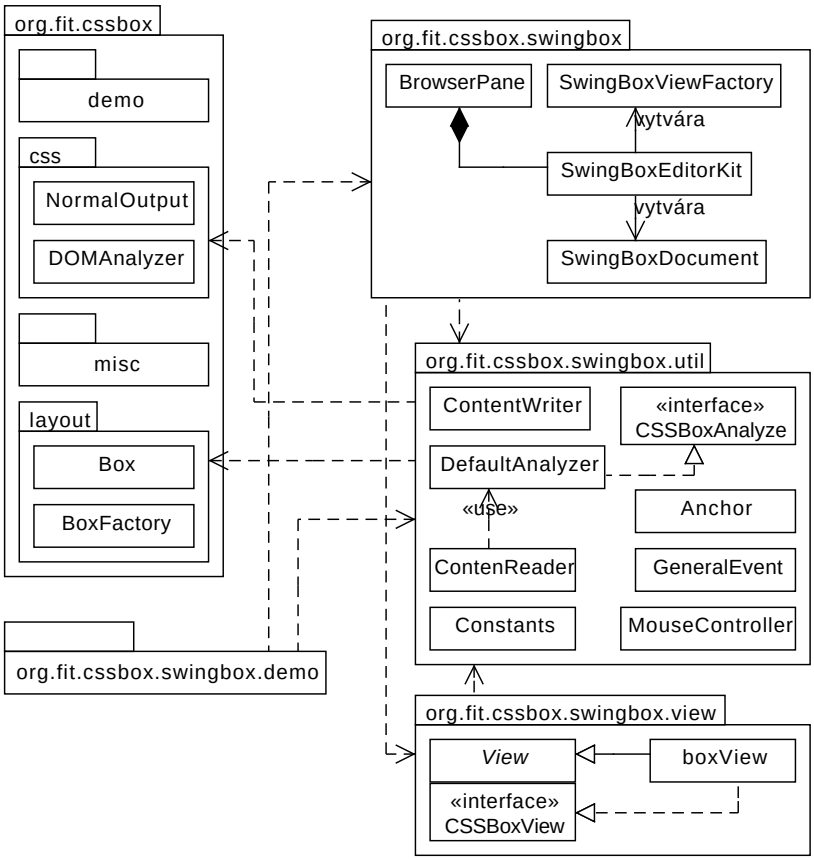
Ako vidno na obrázku 4.1, v aktuálnej implementácií je hierarchia rozdelená nasledovne:

org.fit.cssbox.swingbox - balíček, v ktorom sa nachádzajú dôležité triedy potrebné pre správnu funkčnosť komponentu BrowserPane.

org.fit.cssbox.swingbox.view - združuje všetky triedy a rozhrania súvisiace s vykresľovaním obsahu.

org.fit.cssbox.swingbox.util - súbor znovu použiteľných a pomocných tried a rozhraní.

org.fit.cssbox.swingbox.demo - balíček obsahujúci demonštračnú aplikáciu.



Obrázek 4.1: Hierarchia balíčkov tried.

4.2 SwingBox

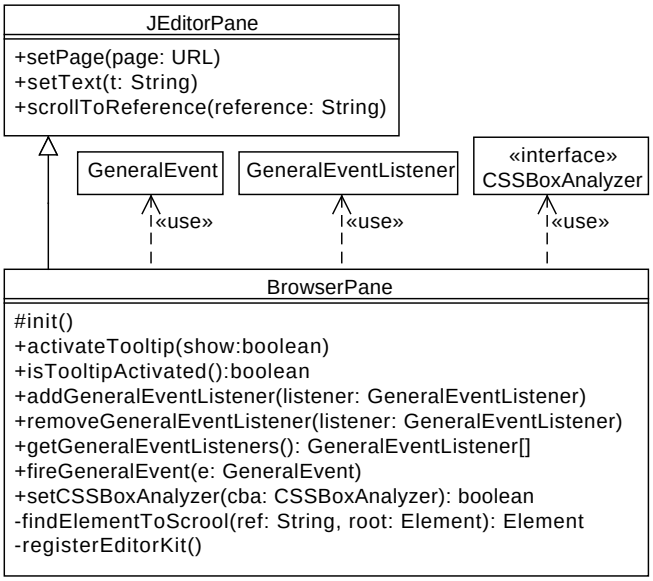
4.2.1 BrowserPane

Je hlavným komponentom užívateľského rozhrania, odvodený od JEditorPane, ktorý zastrešuje celé dianie spracovania a zobrazovania obsahu (nadnesene ho môžeme považovať za „rozhranie“ medzi vonkajším a vnútorným svetom).
Podobne ako v [6], sa počas vytvárania inštancie registruje typ obsahu (MIME[4]) spolu s `ClassLoaderom`[5] a plným, kvalifikovaným menom triedy, ktorá implementuje EditorKit pre spracovanie tohto obsahu. Volanie metódy `registerEditorKitForContenType` zabezpečuje registráciu, pričom SwingBoxEditorKit je automaticky registrovaný s typmi obsahu `text/html`, `text/xhtml` a `application/xhtml+xml`. Prípadné doplnenie registrácie o ďalšie typy obsahu, je otázkou volania jedného riadku kódu. Ak BrowserPane je vyzvaný na spracovanie obsahu, prejde si svoje záznamy či obsahuje požadovaný EditorKit. Ak sa taký nájde, pomocou registrovaného ClassLoadera je dynamicky vytvorený objekt EditorKitu, ktorému sú predané dáta na spracovanie. Ak nie je nájdený vhodný EditorKit, je použitý interný PlainEditorKit. BrowserPane je od začiatku nastavený ako *needitovateľný* (`setEditable(false)`) z dôvodu funkčnosti spracovania odkazov. Ak by hodnota bola `true`, tak pri kliknutí by sa nastavila nová pozícia pre kurzor (treba pamätať, že JEditorPane je univerzálny, textovo orientovaný komponent). Počas inicializácie je tiež volaná metóda `activateTooltip(true)`, ktorá zaregistruje BrowserPane v inštancií `ToolTipManagera`[6], vďaka ktorému sa budú zobrazovať tooltip popisky.
Komponent generuje udalosti typu `GeneralEvent` (udalosti všeobecného charakteru ako napr. začiatok načítavania stránky), ktoré zasiela na spracovanie do `GeneralEventListenerov` a umožňuje nastaviť vlastný `CSSBoxAnalyzer` (užívateľsky definovaný spôsob analýzy a spracovania vstupných dát). Podrobnosti v bode 4.3.4, resp. v bode 4.3.3.

4.2.2 SwingBoxDocument

Dokument vo všeobecnosti je kontajnerom pre obsah a pre textové komponenty swingu predstavuje model. Je navrhnutý tak, aby dokázal uspokojiť potreby od jedoduchých až po komplexné ako je práve HTML.
Dokument dedí od triedy `DefaultStyledDocument`, čo nám zabezpečí prácu so štýlmi a hlavne umožní prácu s `JEditorPane`. Počas vytvárania je v konštruktori volaná metóda `setDocumentFilter(null)`, čím zabezpečíme, že nie je volaný žiaden `DocumentFilter`[7], ktorý by rozhodoval o vkladaní či mazaní textu. Naším cieľom nie je podpora úpravy obsahu, ale len jeho zobrazenie. Pre vloženie obsahu do tohto dokumentu je možné použiť metódu `insertString(int offset, String str, AttributeSet a)`[8]. Prvým parametrom je pozícia, na ktorú sa vkladá text do dokumentu (druhý parameter) a množina atribútov predstavujúca vlastnosti vkladaného textu, ako napr. použitý font a farba. Do `AtributeSetu`[9] je možné pridať i vlastné objekty, ktoré sú potom dostupné v Elemente a následne aj vo View objekte. Toto je naplno využité pri odovzdávaní referencie na `Box`, ktorý ďalej

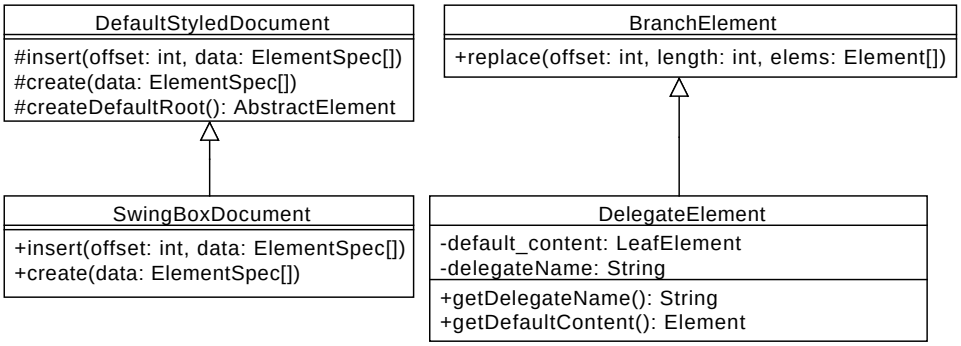
[4] http://www.iana.org/assignments/media-types/index.html
[5] http://download.oracle.com/javase/6/docs/api/java/lang/ClassLoader.html
[6] http://download.oracle.com/javase/6/docs/api/javax/swing/ToolTipManager.html
[7] http://download.oracle.com/javase/6/docs/api/javax/swing/text/DocumentFilter.html
[8] http://download.oracle.com/javase/6/docs/api/javax/swing/text/AbstractDocument.html#insertString(int,java.lang.String,javax.swing.text.AttributeSet)
[9] http://download.oracle.com/javase/6/docs/api/javax/swing/text/AttributeSet.html



Obrázek 4.2: Diagram triedy BrowserPane.

využíva príslušný `View`[10] objekt, ako aj pri objekte `Anchor` pre prácu s odkazmi. V dokumentácií k metóde je uvedené, že dochádza k aktivácií zápisového zámku (vstup do kritickej sekcie) počas vykonávania zmien v dokumente a následne k notifikácií zainteresovaných pozorovateľov. Predpokladajme, že by sme v cykle postupne vkladali reťazce spracovaného obsahu. Je zrejmé, že by dochádzalo k zbytočnej réžií zamykania a odomykania kritickej sekcie a notifikovania pozorovateľov, teda k zbytočnej strate výkonu, čo by sa mohlo prejaviť „mrznutím“ užívateľského rozhrania. Miesto tejto metódy je možné použiť metódu `insert(int offset, ElementSpec[] data)`, resp. `create(ElementSpec[] data)`, ktoré dokážu na jedno zamknutie kritickej sekcie (a jedno volanie zainteresovaných pozorovateľov) zmeniť obsah dokumentu. `ElementSpec`[11] v týchto metódach predstavuje len „objektovú špecifikáciu“ pre dynamické vytváranie konkrétnych elementov pre dokument, čo využíva `ContentReader` (bod 4.3.5). Podľa testov [5] je možné dosiahnúť zrýchlenie spracovania až na úrovni jedného rádu. V testoch je síce použitý JTextPane, ale ten je v skutočnosti odvodený od JEditorPane. Výsledky hlavne závisia od rýchlosti procesora, pamäti, verzie JRE/JDK a v neposlednom rade od velkosti a zložitosti obsahu dokumentu.
Ďalším problémom je, že zmienené metódy sú neverejné a dostupné len ako *protected*. Preto boli v dokumente predefinované ako verejné (*public*). Predefinovaná je aj metóda `createDefaultRoot()`, ktorej úlohou je vytvoriť koreňový element, pre reprezentovanie štruktúry dokumentu. Implementácia vracia inštanciu triedy `DelegateElement`. Do tohto koreňa je dynamicky pripájaný obsah (strom objektov typu Element), ktorý sa postupne začne zobrazovať. Ak nie je nič pripojené, je zobrazený východzí obsah vo forme znaku nového riadku.

[10] http://download.oracle.com/javase/6/docs/api/javax/swing/text/View.html
[11] http://download.oracle.com/javase/6/docs/api/javax/swing/text/DefaultStyledDocument.ElementSpec.html



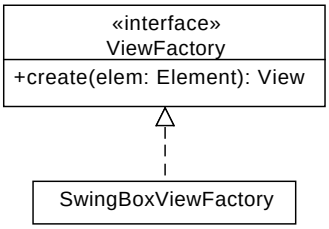
Obrázek 4.3: Diagram triedy SwingBoxDocument a DelegateElement.

DelegateElement

Dôvodom pre vlastný koreňový element je, že v pôvodnej implementácií sa premenil na objekt `BoxView`[12] a nastali problémy s relatívnymi a absolútnymi pozíciami (napr. problém so zvýraznením textu). Navyše, je celkom logické, že viewport, plocha, ktorá zobrazuje výsledný obsah užívateľovi, je umiestnený na čo najvyššej úrovni. Preto je DelegateElement navrhnutý tak, aby obsahoval najviac jedného potomka a delegoval na neho čo najviac funkcionality. V našom prípade je to viewport, prípadne východzí obsah.

4.2.3 SwingBoxViewFactory

Názov prezrádza, že ide o objekt s továrenským správaním sa. Jedinou úlohou implementovaného rozhrania `ViewFactory`[13], je vytvoriť správny View objekt podľa identifikátora v atribútoch elementu. V prípade, že vstupný element nemá nastavený atribút, je použitý výsledok metódy `getName()`. Metóda, okrem View objektov pre SwingBox, je schopná vytvoriť aj štandardné View objekty z balíka `javax.swing.text`, konkrétne *BoxView*, *IconView* ako aj *ComponentView*. V prípade, že metóda nie je schopná vyrobiť požadovaný objekt, je vrátený aspoň *LabelView*, ktorý sa pokúsi zobraziť obsah daného elementu ako jednoduchý text.



Obrázek 4.4: Diagram triedy SwingBoxViewFactory.

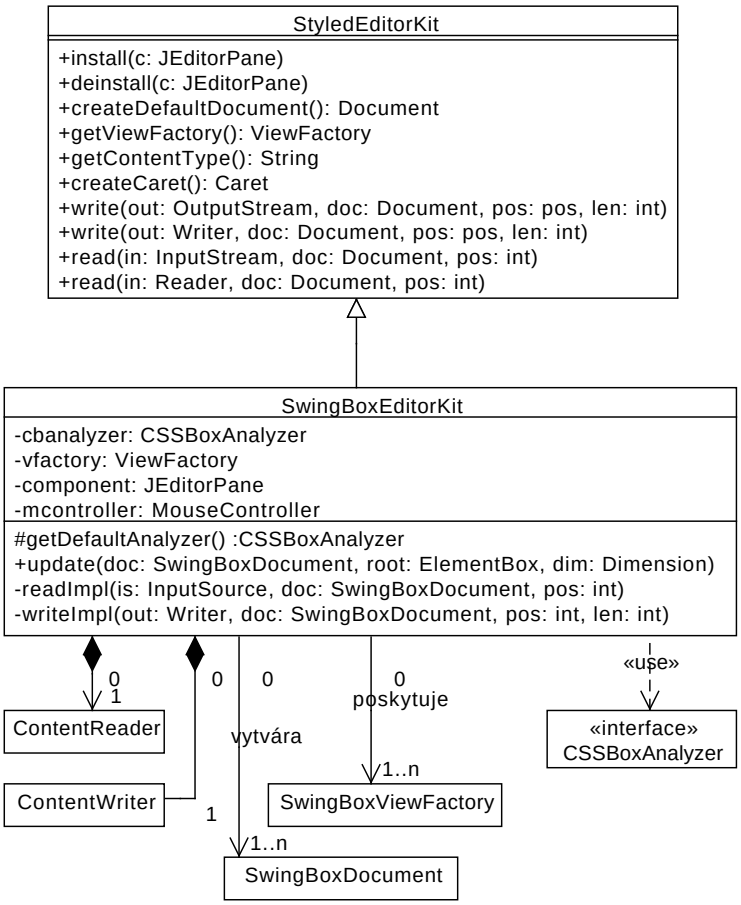
[12] http://download.oracle.com/javase/6/docs/api/javax/swing/text/BoxView.html
[13] http://download.oracle.com/javase/6/docs/api/javax/swing/text/ViewFactory.html

4.2.4 SwingBoxEditorKit

Objekt predstavujúci súbor potrebných vecí, ktoré z textového komponentu spravia rozumne pracujúci editor pre daný typ obsahu. V konštruktori sa zisťuje či vlastnosť prostredia `swingbox.default.analyzer`[14] a `swingbox.document.async_load_priority`[15] je nastavená.
Prvá vlastnosť predstavuje plne kvalifikované meno triedy implementujúcej rozhranie `CSSBoxAnalyzer`. Takto zadaný analyzátor je neskôr pomocou reflexie vytvorený a použitý k spracovaniu vstupných dát. Ak vlastnosť nie je definovaná, je použitá východzia hodnota `org.fit.cssbox.swingbox.util.DefaultAnalyzer`. Ako už bolo spomenuté v bode 4.2.1, CSSBoxAnalyzer je možné zadať aj priamo, volaním metódy `setCSSBoxAnalyzer` nad objektom BrowserPane, ktorý volanie deleguje do nastaveného EditorKitu (za predpokladu, že je inštancia typu SwingBoxEditorKit). Zadanie analyzátora pomocou vlastnosti prostredia, predstavuje alternatívny spôsob, pomocou ktorého je možné ovplyvniť správanie aplikácie pri spúšťaní, bez nutnosti nanovo kompilovať aplikáciu.
Druhá vlastnosť určuje či sa má dokument spracovávať synchrónne, alebo asynchrónne, teda paralelne. Hodnota predstavuje číslo. Ak je menšie ako nula, dokument sa spracuje synchrónne, inak udáva prioritu vlákna (v zmysle triedy `Thread`[16]). Ak hodnota nie je zadaná, dokument sa spracuje synchrónne.
Počas inštalácie EditorKitu do inštancie BrowserPaneu sa uchováva referencia na objekt do ktorého sa inštaluje a registruje sa `MouseController`, ktorý sa stará o zmenu kurzora myši pri prejdení cez odkaz, ako aj o generovanie udalostí pri kliknutí na odkaz. V opačnom zmysle je vykonávaná deinštalácia, ktorá odstráni všetky registrované objekty z inštalácie. Takto je zaručené, že v pamäti neostanú zbytočné referencie, ktoré by bránili odstráneniu objektov GarbageCollectorom. Volanie metódy `createDefaultDocument`, zabezpečí vytvorenie vždy novej inštancie `SwingBoxDocumentu`. V tejto metóde sa nastavuje priorita pre spracovanie dokumentu. Metóda `getViewFactory` vždy vracia rovnakú inštanciu `SwingBoxViewFactory`, ktorá je vytvorená až pri prvom volaní tejto metódy. Úlohou `getContentType` je vrátiť označenie typu obsahu, ktoré EditorKit spracováva. V našom prípade je to reťazec `text/html`. `CreateCaret` vždy vracia hodnotu `null`, pretože nepotrebujeme žiaden cursor na vizuálne označenie miesta pre vloženie obsahu – editácia nie je podporovaná. EditorKit obsahuje dve metódy `write`, ktoré sa líšia len parametrom pre výstupný prúd dát. V jednej metóde to je parameter pre znakovo orientovaný prúd (`Writer`[17]) a v druhej pre bajtovo orientovaný prúd (`OutputStream`[18]). Obe metódy sú zjednotené v metóde `writeImpl`, ktorá pre zápis dát používa Writer. V tejto metóde je premenený aktuálny obsah dokumentu na zdrojový kód vo forme HTML a CSS pomocou triedy `ContentWriter` tak, aby čo najlepšie odzrkadľoval skutočnosť. Metóda write je volaná napr. metódou `getText()` v triede JEditorPane. Podobne je spravená i metóda `read`, ktorá tiež existuje v dvoch verziách pre `Reader`[19] a `InputStream`[20] a obe využívajú metódu `readImpl`. Táto metóda spracuje obsah vo vstupnom prúde dát pomocou triedy `ContentReader` a triedy implementujúcej `CSSBoxAnalyzer`. Pre správne rozloženie obsahu na obrazovke, je potrebné vedieť aspoň približné rozmery zobrazovacej plochy. Tento údaj dostaneme z odloženej referencie

[14] definované ako Constants.DEFAULT_ANALYZER_PROPERTY
[15] definované ako Constants.DOCUMENT_ASYNCHRONOUS_LOAD_PRIORITY_PROPERTY
[16] http://download.oracle.com/javase/6/docs/api/java/lang/Thread.html
[17] http://download.oracle.com/javase/6/docs/api/java/io/Writer.html
[18] http://download.oracle.com/javase/6/docs/api/java/io/OutputStream.html
[19] http://download.oracle.com/javase/6/docs/api/java/io/Reader.html
[20] http://download.oracle.com/javase/6/docs/api/java/io/InputStream.html

na komponent, získanej pri inštalácií EditorKitu. Ak je komponent uložený v `JScrollPane`, použijeme `getExtentSize()`, inak `getBounds().getSize()` komponentu. Ak získané rozmery sú malé (menšie ako 10), tak sú použité rozmery aktuálnej obrazovky z `Toolkitu`, predelené hodnotou 2.5 (táto hodnota bola zvolená experimentálne). Následne je volaná metóda read z triedy ContentReader, ktorej výsledkom je zoznam elementov definovaných ako `ElementSpec`. Tento zoznam je transformovaný na pole a zapísaný do dokumentu použitím metódy `create`. Metódu read v EditorKite volá napr. `setPage`. Problémom pri spracovaní sú znakové sady. Ak server neuvedie použitú znakovú sadu v http hlavičke, tak JEditorPane pri spracovaní vstupu zo socketu zle premieňa bajty na znaky, čo má za následok zobrazovanie štvorčekov či iných nesprávnych znakov (napr. http://www.sme.sk). Za tento jav je zodpovedný `InputStreamReader`, ktorý sa stará o konverziu dát medzi streamom a readerom a ako východzie kódovanie znakov použie UTF-8. Aby sa tomuto javu zabránilo, je potrebné reimplementovať metódu `getStream` z triedy JEditorPane. Jedným z možných riešení je analyzovať HTML hlavičku a nájsť použité kódovanie (za predpokladu, že je tam skutočne uvedené), alebo podľa domény najvyššej úrovne špekulatívne určiť. V aktuálnej implementácií tento problém nie je riešený, nakolko nevznikol vlastným zavinením.

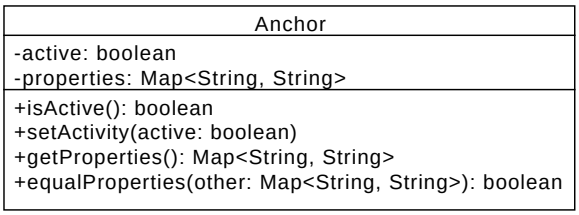


Obrázek 4.5: Diagram triedy SwingBoxEditorKit.

4.3 Pomocné objekty

4.3.1 Anchor

Ide o jednoduchú, pomocnú triedu pre prácu s odkazmi. Každý element má pridelenú vlastnú inštanciu tejto triedy, pričom View objekty ju zdielajú so svojím Elementom. Trieda obsahuje členskú premennú `active`, ktorá určuje či objekt, ktorému patrí táto inštancia je odkazom, alebo nie. Ďalej obsahuje hash mapu, ktorá združuje informácie o odkaze. Momentálne sú podporované atribúty `href`, `name`, `title` a `target`. Ak počas vytvárania View objektov, objekt zistí, že názov tagu, ktorý predstavuje pridelený Box, je rovný `a` (teda *anchor*), nastaví active na true a zistí hodnoty atribútov. Ak objekt zistí, že jeho rodič je odkaz, skopíruje si všetky jeho vlastnosti uložené v hash mape a nastaví active na true.



Obrázek 4.6: Diagram triedy Anchor.

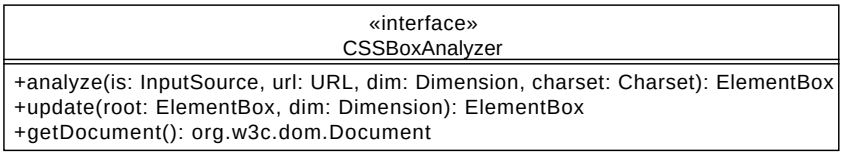
4.3.2 Constants

Constants je trieda vo forme knižnice, ktorej jedinou úlohou je na jednom mieste koncentrovať všetky konštanty globálneho charakteru použité vo SwingBoxe. Sú tu definované konštanty pre identifikáciu vlastnosti prostredia (používa `EditorKit`), konštanty použité na definovanie atribútov elementu (využíva `ContentReader`), identifikátory elementov, podľa ktorých sú vytvárané View objekty a identifikátory hodnôt pre odkazy (tieto používa `Anchor`). Pri vytváraní mien pre konštanty bola snaha, aby daná skupina používala nejaký *vzorec*. Vlastnosti prostredia končia slovom `PROPERTY` (až na `PROPERTY_NOT_SET`), atribúty elementov začínajú slovom `ATTRIBUTE`, identifikátor elementov má príponu `BOX` a identifikátory hodnôt pre odkazy sú vo formáte:
`ELEMENT_`*názov HTML elementu*`_ATTRIBUTE_`*názov HTML atribútu*.

4.3.3 CSSBoxAnalyzer

Hlavnou úlohou tohoto rozhrania je oddelenie SwingBoxu od kódu CSSBoxu, ktorý sa môže meniť, prípadne je želané predspracovanie ešte pred spracovaním vstupu CSSBoxom. Rozhranie pozostáva z troch metód, pričom najdôležitejšou je `analyze`. Na vstupe obdrží obsah na spracovanie vo forme dátového toku, URL zdroja dát, rozmery cieľovej plochy pre rozloženie obsahu a znakovú sadu (ak je dostupná). Výstupom je inštancia typu `ElementBox` predstavujúca koreň celej hierarchie Boxov. V prípade problémov so spracovaním je vyhodená všeobecná výnimka Exception. Metóda `update` je veľmi podobná `analyze`. Jej úlohou je pre zadaný nový rozmer zobrazovacej plochy a pre inštanciu typu ElementBox, prepočítať rozloženie obsahu bez opätovného spracovania vstupných dát. Táto metóda by mala prepočet vykonať čo najrýchlejšie, inak sa môže prejavovať *„zamŕzanie“* užívateľského

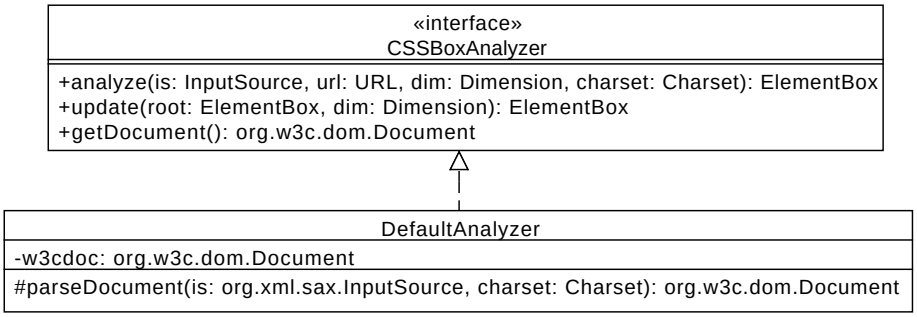
rozhrania. V prípade problémov, je vyhodená výnimka Exception. Poslednou metódou je `getDocument`, ktorá by mala vrátiť dokument v zmysle `org.w3c.dom.Document`[21], ktorý by mal vzniknúť počas spracovania obsahu metódou analyze (produkt použitia DOMParsera). Výstup metódy používa SwingBoxEditorKit v metóde `write`, na premenu zobrazeného obsahu na zdrojové dáta.



Obrázek 4.7: Diagram rozhrania CSSBoxAnalyzer.

DefaultAnalyzer

Jednoduchá, východzia implementácia vyššie spomenutého rozhrania, ktorá je implementovaná tak, aby bolo možné z tejto triedy dediť a zasahovať do spracovania podľa vlastných predstáv. Pridaná je metóda `parseDocument`, ktorá izoluje spracovanie vstupných dát DOMParserom. V tejto implementácií je použitý `Xerces`[22] spolu s `CyberNeko HTML Parser`[23]. Prekonaním tejto metódy je možné dosiahnúť použitie vlastného parsera (za predpokladu, že výstupom bude `org.w3c.dom.Document`). Po získaní dokumentu pomocou parseDocument v metóde analyze, obsah ďalej spracuje `DOMAnalyzer` a `BoxFactory`, ktoré už patria do CSSBoxu. Výsledkom spracovania je inštancia triedy `Viewport`, koreň hierarchie pre výsledné zobrazenie Boxov na obrazovke.



Obrázek 4.8: Diagram triedy DefaultAnalyzer.

4.3.4 GeneralEvent & GeneralEventListener

Počas riešenia zadania, vznikla myšlienka pre signalizáciu rôznych, často menej dôležitých, ale zaujímavých udalostí ako je napr. začiatok načítavania stránky. JEditorPane v aktuálnej implementácií neposkytuje žiaden typ udalosti, ktorý by bol dostatočne všeobecný pre toto

[21] http://download.oracle.com/javase/6/docs/api/org/w3c/dom/Document.html
[22] http://xerces.apache.org/xerces2-j/
[23] http://nekohtml.sourceforge.net/

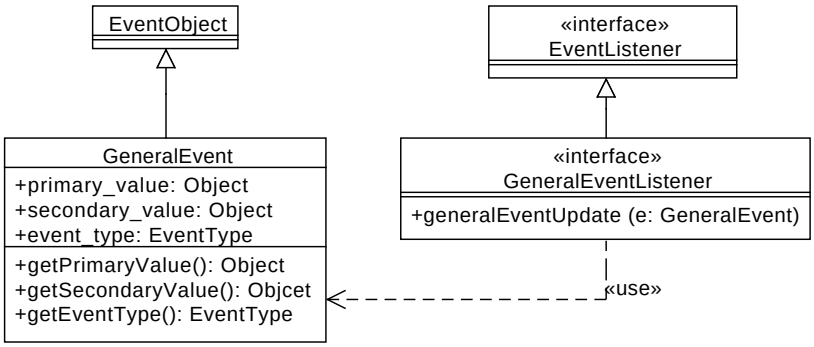
použitie (do úvahy pripadá ešte `PropertyChange`, ale ten má tiež svoje špecifiká).
Udalosť *GeneralEvent* dedí od štandardného objektu `EventObject`[24], pričom v konštruktory je možné definovať objekt od ktorého pochádza udalosť (zdroj), typ vzniknutej udalosti (`EventType`) a dva všeobecné parametre pre obsiahnutie potrebných dát (primary_value a secondary_value). EventType má momentálne definované tieto typy udalostí:

page_loading_begin - signalizácia začiatku načítavania novej stránky. Udalosť vznikne v metóde `setPage` a `setText` v BrowserPane. Parameter primary_value obsahuje URL zdroja dát, ak je známy, inak hodnotu `null`.

page_loading_end - signalizovanie ukončenie načítavania stránky. Udalost vznikne na konci spracovania obsahu SwingBoxEditorKitom v metóde `readImpl`. Je garantované, že udalosť vznikne až `po úspešnom zápise` spracovaných dát do SwingBoxDocumentu, no obsah ešte nemusí byť vykreslený či viditeľný. Parameter primary_value obsahuje URL zdroja dát, ak je známy, inak hodnotu `null`.

page_loading_error - udalosť signalizuje chybu počas spracovania obsahu.

GeneralEventListener je rozhranie, odvodené od rozhrania `EventListener`[25] s jedinou metódou, `generalEventUpdate`, ktorá sa stará o prijatie a spracovanie udalostí. Implementáciou tohto rozhrania a následnou registráciou v BrowserPane pomocou metódy `addGeneralEventListener` je možné zabezpečiť si prijímanie udalostí (samozrejmosťou sú metódy pre odstránenie a získanie zoznamu registrovaných GeneralEventListenerov).
Do budúcna sa predpokladá, že GeneralEvent ako aj GeneralEventListener budú obohatené o ďalšie typy udalostí či možnosti spracovania. Napríklad, po spracovaní stránky, by bolo vhodné, vrátiť zoznam tagov `meta` v hlavičke HTML dokumentu.



Obrázek 4.9: Diagram triedy GeneralEvent a GeneralEventListener.

4.3.5 ContentReader

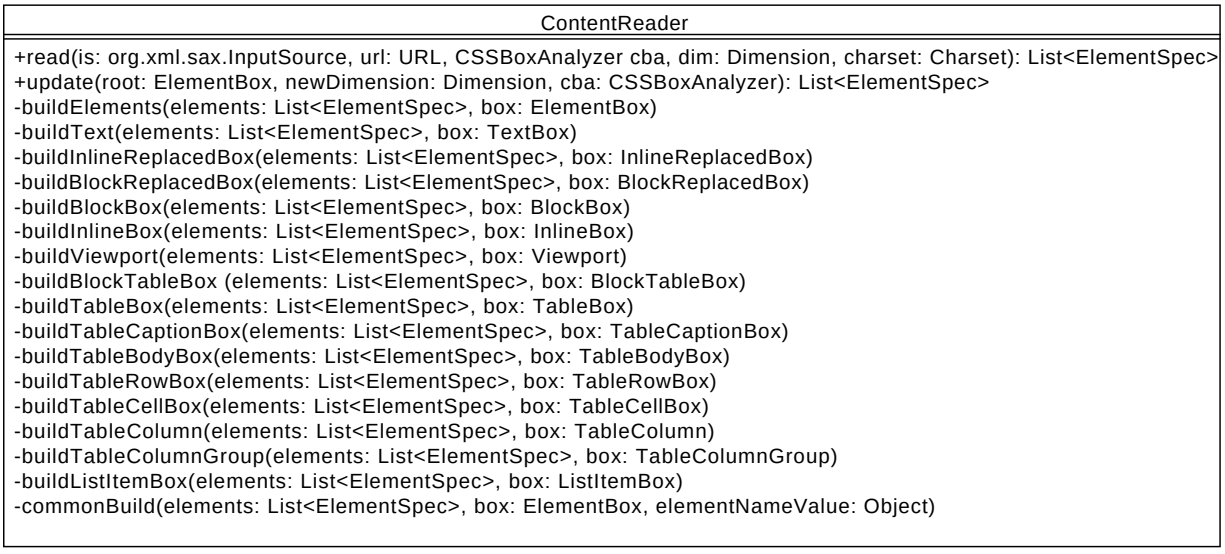
Je miestom, kde dochádza k premene vstupných dát, ktoré sú ešte vo vstupnom prúde, na reprezentáciu pomocou Box objektov, ktoré sú transformované na výsledný zoznam objektov `ElementSpec` popisujúcich elementy pre SwingBoxDocument. To všetko sa odohráva v metóde `read`, ktorá na vstupe obdrží `InputSource`[26] (pomocná trieda používaná

[24] http://download.oracle.com/javase/6/docs/api/java/util/EventObject.html
[25] http://download.oracle.com/javase/6/docs/api/java/util/EventListener.html
[26] http://download.oracle.com/javase/6/docs/api/org/xml/sax/InputSource.html

XML parsermi pre unifikáciu zdroja dát – InputStream a Reader), URL zdroja dát, inštanciu implementujúcu rozhranie CSSBoxAnalyzer, (približné) rozmery zobrazovacej plochy pre rozloženie obsahu a použitú znakovú sadu (ak je známa). Následne je predané riadenie metódou `analyze` s príslušnými parametrami do inštancie implementujúcej rozhranie `CSSBoxAnalyzer`. Výsledkom je inštancia `Viewport`, predstavujúca koreň hierarchie spracovaných Boxov. Ďalej je volaná metóda `buildViewport` (prípadne `buildElement`, ak by výsledkom nebola inštancia typu Viewport). Táto metóda, podobne ako ostatné (v tvare build a názov boxu, napr. buildListItemBox) sa starajú o vytvorenie špecifikácie elementu pre SwingBoxDocument pomocou `ElementSpec`. Pre definovanie Elementu, je nutné:

1. Vytvoriť inštanciu ElementSpecu s parametrom *ElementSpec.StartTagType*.
2. Vytvoriť inštanciu ElementSpecu s parametrom *ElementSpec.ContentType*, spoločne s polom znakov, offsetom a dĺžkou obsahu v tomto poli.
3. Vytvoriť inštanciu ElementSpecu s parametrom *ElementSpec.EndTagType*.

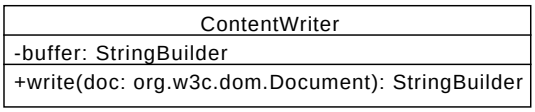
Každý ElementSpec má ešte parameter pre zadanie množiny atribútov, ktorá by mala byť rovnaká aspoň pre StartTagType a EndTagType. Tieto atribúty bude mať k dispozícií Element a neskôr aj View objekt. Práve v tomto mieste je vytváraná osobitná inštancia triedy `Anchor` pre každý budúci Element, predáva sa referencia na príslušný `Box`, nastavuje sa identifikátor, podľa ktorého `SwingBoxViewFactory` vytvorí príslušný View objekt, apod. Toto predstavuje nutné minimum, ktoré musia mať Elementy definované vo svojich atribútoch pre správnu funkčnosť, čo pre väčšinu z nich postačuje. Aby sa predišlo duplicite kódu, je za týmto účelom vytvorená metóda commonBuild, ktorá sa stará o spomenuté veci. Boxy typu *TextBox*, *InlineReplacedBox* a *BlockReplacedBox* obohacujú množinu atribútov o ďalšie, ktoré sa vzťahujú k reprezentácií ich obsahu. Do budúcna sa počíta, že View objekty (ako aj Elementy) sa budú správať viac dynamicky, pričom v atribútoch budú aktuálne hodnoty reprezentované užívateľovi.



Obrázek 4.10: Diagram triedy ContentReader.

4.3.6 ContentWriter

Túto triedu využíva SwingBoxEditorKit pri volaní metódy `write` na premenu aktuálneho obsahu dokumentu na textovú reprezentáciu. Na dosiahnutie tohoto sa používa trieda `NormalOutput` z prostredia CSSBoxu a jej metóda `dumpTo`. Výstupom je text zdrojového kódu HTML a CSS.



Obrázek 4.11: Diagram triedy ContentWriter.

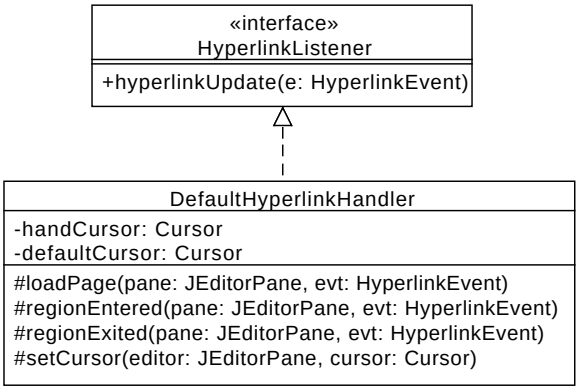
4.3.7 DefaultHyperlinkHandler

Jednoduchá trieda, implementujúca rozhranie `HyperlinkListener`[27], pre spracovanie vygenerovaných udalostí súvisiacich s odkazmi. V metóde `hypelinkUpdate` sa spracovávajú prijaté udalosti. Ak vzniknutá udalosť je typu `HyperlinkEvent.EventType.ACTIVATED`, nastaví sa kurzor myši na východzí a požadovaná stránka sa spracuje metódou `loadPage`. V tejto metóde sa nastaví požadovaná URL na spracovanie metódou `setPage` nad inštanciou typu `JEditorPane`. Ak je udalosť typu `HyperlinkEvent.EventType.ENTERED`, teda kurzor bol umiestnený nad odkaz, dôjde k zmene kurzoru myši na kurzor *„ručičky“*, známy z internetových prehliadačov. V prípade typu udalosti `HyperlinkEvent.EventType.EXITED`, teda kurzor bol premiestnený mimo odkaz, dôjde k nastaveniu kurzoru na východziu hodnotu.
Trieda je navrhnutá tak, aby bolo možné pomocou dedenia upraviť priebeh spracovania udalostí podľa vlastných predstáv (metódy *loadPage*, *regionEntered*, *regionExited*, *setCursor*), napr. dotazovať sa užívateľa či si praje prejsť na danú URL. Navyše, použitie triedy nie je viazané len na BrowserPane, ale môže byť bez zmeny kódu využité v každej triede, ktorá dedí od JEditorPane. Pre správnu funkčnosť, treba inštanciu triedy explicitne registrovať pre príjem udalostí.

4.3.8 MouseController

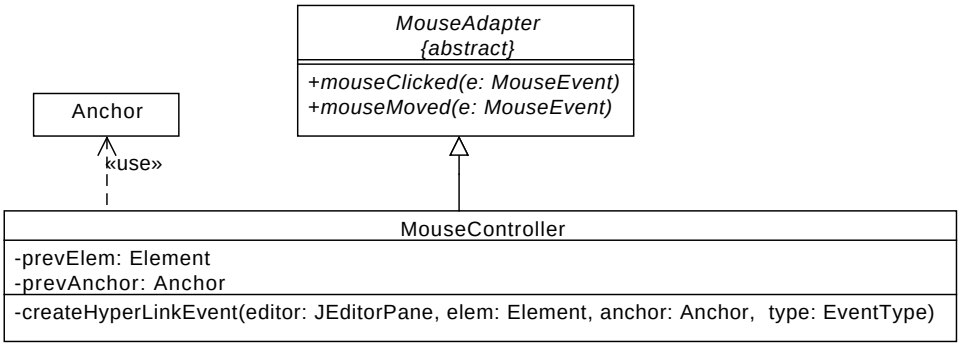
MouseController je trieda zodpovedná za generovanie `HyperLinkEvent`[28] udalostí, ktoré následne spracuje registrovaný objekt implementujúci rozhranie `HyperlinkListener`. To čo nás zaujíma je pozícia, na ktorej užívateľ klikol myšou a pozícia, na ktorú sa presunul. Preto trieda dedí od triedy `MouseAdapter` a implementuje metódy `mouseClicked` a `mouseMoved`. V prvej menovanej metóde, sa získa objekt v ktorom vznikla táto udalosť (inštancia typu JEditorPane, skrátene editor). Následne sa overí, či sa jedná o kliknutie ľavým tlačidlom myši a či editor nie je editovateľný (inak by to mohlo znamenať, že užívateľ len premiestnil kurzor, teda Caret, na inú pozíciu). Potom je zistená presná pozícia (X a Y), kde vznikla udalosť a je vyvolaná metóda `viewToModel` s parametrom tejto pozície nad získaným editorom. Metóda `viewToModel` je trenasformácia medzi súradnicovým systémom View objektov a pozíciami elemtov v dokumente SwingBoxDocument. Vďaka nájdenej pozícií sme

[27] http://download.oracle.com/javase/6/docs/api/javax/swing/event/HyperlinkListener.html
[28] http://download.oracle.com/javase/6/docs/api/javax/swing/event/HyperlinkEvent.html



Obrázek 4.12: Diagram triedy DefaultHyperlinkHandler.

schopný z dokumentu získať konkrétnu inštanciu elementu, ktorý sa tam nachádza. Potom je už jednoduché z množiny atribútov elementu, získať referenciu na pridelený objekt `Anchor`, ktorý už obsahuje konkrétne informácie o odkaze, ktoré nás zaujímajú. Za predpokladu, že `Anchor` skutočne predstavuje odkaz (`isActive() == true`), je zavolaná metóda `createHyperLinkEvent`, ktorá sa už postará o vytvorenie a vyvolanie udalosti.
Metóda `mouseMoved` sa správa podobne, ako predošlá. Na základe aktuálnej pozície kurzora myši sa získa pozícia elementu v dokumente volaním metódy `viewToModel`. Po získaní inštancie elementu podľa pozície v dokumente, sa použije `Anchor`, obsiahnutý v množine atribútov elementu, k porovnaniu s `Anchorom` z predošlého vyvolania udalosti `mouseMoved`. Ak nový `Anchor` má rozdielne vlastnosti (teda rozdielne *href*, *name*, *title*, alebo *target*), vygeneruje sa metódou `createHyperLinkEvent` udalosť `typu EventType.EXITED` nad starým `Anchorom` a následne udalosť typu `EventType.ENTERED` nad novým `Anchorom`. Ak sú vlastnosti rovnaké, nie je vygenerovaná žiadna udalosť.

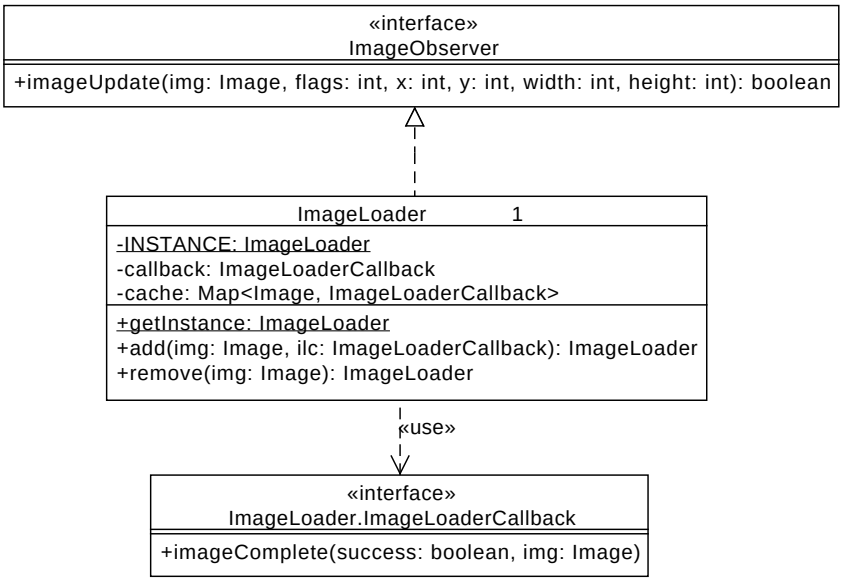


Obrázek 4.13: Diagram triedy MouseController.

4.3.9 ImageLoader

Jednoduchá trieda, ktorej úlohou je asynchrónne načítať na pozadí obrázky a pri úspešnom dokončení, alebo chybe, informovať zadávateľa. K tomuto účelu je v triede definované

rozhranie `ImageLoaderCallback` s metódou `imageComplete`, ktoré informuje o úspechu či neúspechu a odovzdá referenciu na spracovaný obrázok. Pre získanie informácie o stave spracovania obrázku, trieda implementuje rozhranie `ImageObserver`[29], v ktorom sa kontroluje či sú dostupné všetky dáta, alebo došlo k chybe. Trieda je navrhnutá ako singleton a poskytuje metódy pre zadanie a odstránenie obrázka zo spracovania.



Obrázek 4.14: Diagram triedy ImageLoader.

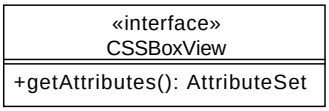
4.4 View objekty

View je veľmi dôležitou triedou balíčka (javax.swing.text), ktorá [7] reprezentuje pohľad na model, teda je zodpovedná za výzor textového komponentu. View je od počiatku navrhnutý ako ľahký prvok (v zmysle použitých prostriedkov). Obsahuje referenciu na rodiča, ktorý môže dodať množstvo vecí, bez nutnosti uchovávania vnútorného stavu a referenciu na časť modelu, teda Element. View nemusí nutne zodpovedať elementu z modelu, ale je bežné, že danému elementu zodpovedá jeden objekt view.

4.4.1 CSSBoxView

Toto rozhranie bolo navrhnuté a implementované každej triede View, hlavne ako identifikátor, rozlišujúci „naše“ View objekty od iných. Vedľajším účinkom je možnosť mať objekty bez rozdielu v jednej kolekcii. Vzhľadom na to, že View objekty využívajú pridelenú inštanciu Boxu, toto rozhranie je zárukou garancie, že v atribútoch objektu (Element aj View) sa referencia na túto inštanciu nachádza. Rozhranie obsahuje metódu `getAttributes`, ktorú objekty už majú zdedenú od `javax.swing.text.View`. Je tu uvedená len pre lepšiu čitateľnosť kódu a zbavenie sa zbytočného pretypovania v pomocných metódach pre získanie inštancie Boxu.

[29] http://download.oracle.com/javase/6/docs/api/java/awt/image/ImageObserver.html



Obrázek 4.15: Diagram rozhrania CSSBoxView.

4.4.2 RootView

Nenápadný, ale veľmi dôležitý View objekt ukrytý v `javax.swing.plaf.basic.BasicTextUI$RootView`, ktorý na jednej strane spája komponent a na strane druhej hierarchiu View objektov. RootView je implementovaný ako potomok abstraktnej triedy View a obsahuje najviac jedného potomka na ktorého deleguje všetky volania. Opačné volania RootView deleguje na `JTextComponent` (napr. preferenceChanged, getDocument). Vykresľovanie je redukované na jednoduché volanie metódy `paint` prideleného potomka, no pred ním je vždy volaná metóda `setSize`. Tímto volaním je potomok donútený upraviť svoje rozloženie, ak od posledného volania sa zmenili rozmery. Volania metódy setSize sú vo SwingBoxe ignorované, pretože zmena rozloženia obsahu je realizovaná globálne, teda obsah so starými rozmermi je kompletne nahradený obsahom prispôsobeným pre nové rozmery. Toto má na starosti `ViewportView`, ktorý bude popísaný neskôr.

4.4.3 ElementBoxView

Základná trieda z ktorej sú odvodené všetky View objekty použité vo SwingBoxe, okrem `TextBoxView`. Ako už bolo spomenuté, trieda implementuje rozhranie `ImageLoaderCallback` a `CSSBoxView` z triedy `ImageLoader` (bod 4.3.9). Trieda dedí od abstraktnej triedy `CompositeView`[30], ktorá je navrhnutá pre správu malého množstva View potomkov. Triedu je možné použiť ako odrazový mostík pre implementáciu vlastných View objektov správajúcich sa ako kolekcie (ako napr. štandardný `BoxView`).
Za jediný parameter konštruktora je dosadená inštancia typu Element, ktorá je delegovaná do nadtriedy pre ďalšie spracovanie. Ďalej sa nastavuje smer hlavnej osi, teda predpokladaný smer expanzie objektu po vizuálnej stránke. Predpokladaný je vertikálny smer, no v prípade inštancie triedy BlockBoxView s nastavenou CSS vlastnosťou float[31], je to horizontálny smer. Z množiny pridelených atribútov je získaná inštancia triedy `Anchor` a `Box`. V prípade problémov s inštanciami je vyvolaná výnimka `IllegalArgumentException`. Následne sú v metóde `loadElementAttributes` načítané informácie o odkaze, ak ho pridelený Box priamo reprezentuje a uchované v objekte Anchor. Metóda `loadBackgroundImage` má za úlohu na pozadí získať obrázok `background-image` (za pomoci ImageLoadera), ak je uvedený v CSS prideleného Boxu. Tu nastáva problém, pretože hodnota tejto vlastnosti je často udávaná ako relatívne URI k umiestneniu súboru v ktorom je definované a dokument si pamätá len URL, ktoré bolo použité pre získanie stránky. Preto, po pokuse o získanie absolútneho umiestnenia, nie sme schopný získať URI s platným umiestnením obrázka. Napríklad, namiesto `http://www.phoronix.com/phxcms7-css/phoronix-header.png` je výsledkom `http://www.phoronix.com/phoronix-header.png`, čo nie je platné umiestnenie. Z tohto dôvodu je metóda považovaná za experimentálnu a vo výslednom kóde nie

[30] http://download.oracle.com/javase/6/docs/api/javax/swing/text/CompositeView.html
[31] http://www.w3schools.com/css/css_float.asp

je volaná. V hierarchií View objektov, má každý svojho rodiča. Priradenie je vykonané prostredníctvom metódy `setParent`[32], ktorá podľa dokumentácie je garantovaná, že bude volaná ako prvá metóda pred ostatnými s parametrom udávajúcim rodiča, alebo ako posledná s hodnotou `null`. Toto je naplno využité pri distribúcií informácie, že rodič je odkazom. Ak ním skutočne je, sú od rodiča prevzaté všetky informácie o odkaze. To sa deje v metóde `pre_setParent`, ktorá musí byť volaná ešte pred delegovaním rodiča v metóde setParent do nadtriedy, inak by hrozilo, že informáciu obdržia len potomkovia prvej úrovne. Metóda setParent volá aj `setPropertiesFromAttributes`, ktorá načíta všetky atribúty do pracovných premenných. Ak iný objekt bude požadovať množinu atribútov tohto objektu a pracovné premenné sa stihli zmeniť, tak množina je nanovo vytvorená. Jedná sa hlavne o tie atribúty, ktoré boli zadané pri vytváraní Elementu.
Pre vykreslenie obsahu sa najskôr nastavia vlastnosti grafického kontextu pomocou metódy `updateGraphics` objektu `VisualContext`, získaného z Boxu. Tu sa nastavujú veci ako *font*, *farba* či *antialiasing*. Následne je vykreslené pozadie a v cykle sa vykresľujú potomkovia pomocou metódy `paintChild`, ktorú je možné predefinovať a prispôsobiť vlastným potrebám.

4.4.4 BlockBoxView

Podľa definície v CSS [1] je *„block box“* boxom blokovej úrovne (zúčastňuje sa formátovania blokového kontextu) a súčasne je blokovým kontajnerom (buď obsahuje boxy len blokovej úrovne, alebo obsahuje len elementy riadkovej úrovne). BlockBoxView dedí od `ElementBoxView`. Upravená je implementácia metódy `isVisible` tak, aby reflektovala viditeľnosť prideleného Boxu. K funkčnosti je pridaná implementácia CSS vlastnosti `overflow` [1], ktorá špecifikuje čo sa má stať, ak obsah presahuje vymedzený priestor. Funkčnosť je obmedzená len na ponechanie obsahu (hodnota *visible*), alebo na useknutie obsahu (ostatné hodnoty). Hodnota *scroll* je technická výzva, pretože k obsahu je pridaný aj posúvací mechanizmus, vďaka ktorému je dostupný celý obsah. Teoreticky je toto možné s použitím triedy `ComponentView` spolu s `JScrollPane` a `JComponentu`, ktorý by vykresloval obsah. Vzhľadom na komplikovanosť tohto riešenia, ktoré je nad rámec práce, nie je implementované.

4.4.5 InlineBoxView

Táto trieda je veľmi podobná predošlej a rovnako dedí od `ElementBoxView`. Jej hlavnou úlohou je [1] horizontálne zoraďovať objekty, ktoré obsahuje, jeden za druhým, od počiatku bloku v ktorom sa nachádza. V konštruktori je volaná metóda `setAxis`, ktorá nastavuje hlavnú os na horizontálny smer a definuje si vlastnú interpretáciu viditeľnosti podľa priradeného Boxu. Vykresľovanie je realizované v rovnakom duchu ako má `BlockBoxView`.

4.4.6 ViewportView

Trieda predstavuje *„priezor“*, ktorým je možné pozorovať obsah. Implementácia je riešená ako špeciálny typ `BlockBoxView`, pričom sa stará aj o detekciu zmien rozmerov zobrazovacej plochy a inicializuje prepočet rozloženia obsahu pre nové rozmery. K tomuto účelu trieda

[32] http://download.oracle.com/javase/6/docs/api/javax/swing/text/View.html#setParent(javax.swing.text.View)

implementuje rozhranie `ComponentListener`[33] (metódu `componentResized`), ktoré je registrované na rodiča BrowserPanu, ktorý by mal byť inštancie typu `JViewport` (napr. JScrollPane používa JViewport pre zobrazenie obsahu). Registrovanie i odregistrovanie prebieha počas prideľovania rodičov View objektom metódou `setParent`, pri ktorom je zapamätaná referencia na tohto rodiča (vo forme slabej referencie) pre neskoršie použitie. Spracovanie zmien rozmerov sa deje v metóde `checkSize`, ktorá na vstupe obdrží rozmery viditeľnej časti plochy (`getExtentSize`). Ak ani jeden z rozmerov nie je rovný nule, porovná sa či ide o rozmery rôzne od súčasného stavu, pričom šírka musí byť väčšia ako minimálna šírka boxu (`getMinimalWidth`) a súčasne menšia ako maximálna šírka boxu (`getMaximalWidth`). Ak sú podmienky splnené, je volaná metóda `doLayout` s parametrom inštancie SwingBoxDocumentu a novými rozmermi. Tu sa volá metóda `update` zo SwingBoxEditorKitu, ktorá vytvorí `ContentReader` a volá rovnomennú metódu `update`, ktorá predáva spracovanie do inštancie `CSSBoxAnalyzera`. Ten s pomocou CSSBoxu spracuje rozloženie obsahu pre nové rozmery. Stáva sa, že CSSBox neprispôsobí rozloženie novým rozmerom, ak sú menšie ako súčasný stav... V prípade, ak sú definované pevné rozmery, rozloženie sa nemení. Výsledkom `ContentReadera` je zoznam elementov, ktorý je zapísaný do dokumentu metódou `create`, ktorá pôvodný obsah odstráni a nahradí novým.

4.4.7 DelegateView

Implementácia triedy View, ktorá zodpovedá `DelegateElementu`. Ako už bolo spomenuté v bode 4.2.2, úlohou je nahradiť pôvodnú implementáciu, predstavujúcu koreň dokumentu (metóda `createDefaultRoot` v triede SwingBoxDocument). Trieda je navrhnutá tak, aby vždy obsahovala najviac jedného potomka, na ktorého deleguje čo najviac funkcionality, prípadne deleguje na nadriadený View objekt (v tomto prípade `RootView`).

4.4.8 TextBoxView

Táto trieda zodpovedá za reprezentáciu textu s danými vlastnosťami v grafickom kontexte. Trieda je implementovaná ako potomok abstraktnej triedy `View` a implementuje rozhranie `CSSBoxView`. V konštruktori je z množiny atribútov získaná inštancia triedy `Anchor` a `Box`. V prípade problémov je vyvolaná výnimka. V tejto triede je rovnako využitá metóda `setParent` na propagáciu odkazov, ako je tomu v `ElementBoxView`. Ak sa zistí, že rodič je odkazom, sú prevzaté všetky informácie o odkaze, ktoré sú neskôr použité v `getToolTipText` pre zobrazenie popisného textu (titulok a URL). V tomto mieste sa tiež volá `setPropertiesFromAttributes`, ktorá nastaví príslušné pracovné premenné z atribútov.
Trieda reimplementuje metódu `modelToView` (metóda na prepočet pozície zo sveta elementov na príslušné súradnice vo svete View objektov, vo forme inštancie typu Shape) s využitím awt objektu `TextLayout`[34], ktorý predstavuje nemennú grafickú reprezentáciu znakových dát. Poskytované sú mnohé funkcie ako napr. polohovanie a posúvanie kurzora, zvýraznenie logické i vizuálne pre text so zmiešanými smermi („bidi“), testovanie, aký znak sa nachádza na konkrétnej grafickej pozícií, metrické informácie či renderovanie. Podľa požadovanej pozície sa najskôr nastaví `TextHitInfo`[35] (pozícia znaku v texte a smer – *ltr*, alebo *rtl*) a následne je volaná metóda `getCaretInfo`, ktorá vracia informácie o kurzore pre dané

[33] http://download.oracle.com/javase/6/docs/api/java/awt/event/ComponentListener.html
[34] http://download.oracle.com/javase/6/docs/api/java/awt/font/TextLayout.html
[35] http://download.oracle.com/javase/6/docs/api/java/awt/font/TextHitInfo.html

miesto, presnejšie pole dvoch čísel typu float. Pre naše potreby potrebujeme prvé číslo, ktoré reprezentuje vzdialenosť od začiatku, pozdĺž línie základnej čiary (baseline), až k miestu prieniku kurzora s touto čiarou. Výsledné súradnice vo forme inštancie typu `Rectangle`[36], sú odvodené zo vstupného parametra predstavujúceho alokáciu a získanej vzdialenosti.
`ViewToModel` je opačnou metódou k vyššie uvedenej a rovnako využíva `TextLayout`. Zo vstupného parametra predstavujúceho alokáciu je pomocou metódy `hitTestChar` získané `TextHitInfo`. Z tohto objektu je volaním `getInsertionIndex` získaná pozícia znaku, ktorý `TextLayout` našiel na zadanej pozícií. Výsledná pozícia je súčtom počiatočnej pozície elementu a získanej pozície. Text sa vykresľuje len ak je viditeľný, čo určuje priradený Box v metóde `isVisible`. Najskôr je spracované „vysvietenie textu“ pomocou inštancie typu `LayeredHighlighter`[37] a jej metódy `paintLayeredHighlights`. Objekt získame z JTextComponentu volaním `getHighlighter`. Od „zvýrazňovača“ sa získa pole typu `Highlight`[38] volaním metódy `getHighlights`. Prvky tohto pola predstavujú počiatočné a koncové pozície fragmentov textu, ktorý treba zvýrazniť. Tieto prvky sú prechádzané v cykle, pričom môžu byť mimo náš rozsah, úplne pokrývať náš rozsah, alebo pokrývajú len časti nášho rozsahu. Samotné vykreslovanie sa odohráva v metóde `renderContent`. Na rozdiel od ostatných View objektov, ktoré na vykresľovanie využívajú Boxy, sa pridelený TextBox nevyužíva a dokonca ani `VisualContext`. Všetky nastavenia (farba, font, antialiasing apod.) a vykresľovanie sa deje vo vlastnej réžií. K tomuto rozhodnutiu prispel aj fakt, že `TextLayout` poskytuje vykresľovanie a množstvo funkcií k obsahu, ktorý reprezentuje, zatiaľ čo TextBox používa štandardnú metódu `drawString`, dostupnú z grafického kontextu. Z dekorácií je podporované *underline*, *overline*, *line-through* ako aj *blink*. K dosiahnutiu posledne menovanej dekorácie, je využitý štandardný `javax.swing.Timer`[39], ktorý v intervale jednej sekundy spúšťa priradený `ActionEvent`, ktorý volá metódu `repaint`. Toto má za následok vyvolanie vykresľovacej rutiny, ktorá dané miesto prekreslí pozadím (v tomto mieste sa už používa rodič TextBoxu).

4.4.9 ReplacedContent & ReplacedImage

ReplacedContent je abstraktná trieda, ktorá predstavuje obsah pre `ReplacedBox` z prostredia CSSBoxu. Poskytuje jednoduché metódy ako nastavenie, alebo zistenie vlastníka tohto obsahu, získanie informácií o rozmeroch a možnosti vykresliť obsah do zadaného grafického kontextu.
ReplacedImage je potomok triedy ReplacedContent, ktorý reprezentuje obrázok ako obsah. V konštruktori je zistené URL obrázka a inicializované načítavanie. V porovnaní s pôvodnou implementáciou, bol zmenený spôsob načítavania. Namiesto `javax.imageio.ImageIO.read` sa používa systémová sada nástrojov (`java.awt.Toolkit`[40]), ktorá podporuje formáty GIF, JPG, PNG a implementuje funkciu vyrovnávacej pamäte, čo šetrí čas v prípade rovnakých obrázkov. Navyše, obrázky dokáže načítavať paralelne, pričom výsledkom už nie je inštancia typu `BufferedImage`[41], ale len `Image`[42]. Ak je trieda dotazovaná na rozmer obrázka, ale ten ešte nie je spracovaný, tak aktuálne vlákno je uspané po dobu 25 ms a v cykle sa znovu dotazuje. To sa opakuje až do chvíle, kým nie je veľkosť známa, alebo nie je signalizo-

[36] http://download.oracle.com/javase/6/docs/api/java/awt/Rectangle.html
[37] http://download.oracle.com/javase/6/docs/api/javax/swing/text/LayeredHighlighter.html
[38] http://download.oracle.com/javase/6/docs/api/javax/swing/text/Highlighter.Highlight.html
[39] http://download.oracle.com/javase/6/docs/api/javax/swing/Timer.html
[40] http://download.oracle.com/javase/6/docs/api/java/awt/Toolkit.html
[41] http://download.oracle.com/javase/6/docs/api/java/awt/image/BufferedImage.html
[42] http://download.oracle.com/javase/6/docs/api/java/awt/Image.html

vaná chyba. V prípade chyby je vrátená východzia veľkosť 20 pixelov. Trieda implementuje rozhranie `ImageObserver`[43], vďaka ktorému je informovaná o stave spracovania obrázku. V prípade, že obrázok obsahuje sekvenciu snímkov, implementácia ich dokáže postupne zobrazovať (napr. animovaný GIF[44]).

4.4.10 InlineReplacedBoxView & BlockReplacedBoxView

Implementácie oboch tried sú rovnaké až na použitú nadtriedu, preto bude opísaná len `BlockReplacedBoxView`. Úlohou tried je obsluhovať a vykresľovať pridelený obsah, ktorým je inštancia typu `ReplacedContent`. V konštruktori je volanie delegované do nadtriedy (`BlockBoxView`), následne sa získa obsah metódou `getContentObj` z Boxu a nakoniec sa volá `loadElementAttributes`, v ktorej sa získajú hodnoty HTML atribútov *alt* a *title* (predpokladá sa HTML tag img), použité pri zobrazení tooltip popisku. Tento popisok zobrazuje najskôr titulok obrázka, potom titulok odkazu a na záver URL, ak sú dostupné informácie o odkaze. Konverzná funkcia `viewToModel` je implementovaná ako jednoduché porovnanie súradníc s aktuálnym umiestnením. Podľa výsledku je vrátená počiatočná, alebo koncová pozícia Elementu. Opačná konverzia, `modelToView`, porovnáva či požadovaná pozícia sa nachádza medzi počiatočnou a koncovou pozíciou Elementu a vráti inštanciu typu Shape. Ak pozícia je mimo rozsah, je vyvolaná výnimka `BadLocationException`. Ak sú dostupné dáta pre obsah, vykresľovanie je delegované do inštancie typu `ReplacedContent`, ktorá už presne vie ako vykresliť obsah. Ak dáta sú nedostupné, je vykreslený aspoň titulok (ak je známy), alebo sa plocha *„preškrtne“* na znak chýbajúceho obsahu.

4.4.11 Ostatné View Objekty

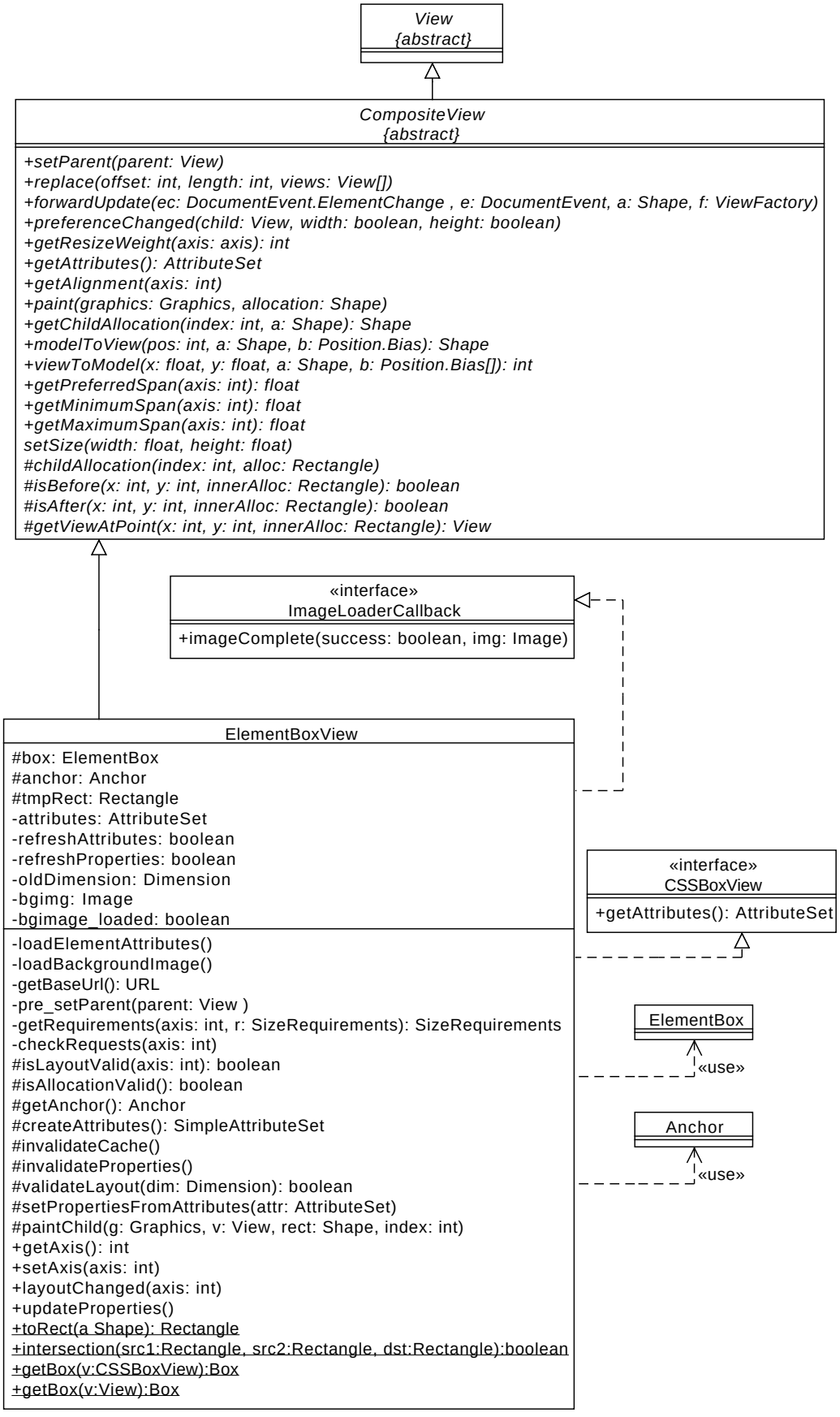
Ostatné implementácie View objektov, menovite *ListItemBoxView*, *TableBodyBoxView*, *TableCaptionBoxView*, *TableCellBoxView*, *TableColumnView*, *TableBoxView*, *TableColumnGroupView* a *TableRowBoxView*, sú potomkami `BlockBoxView`. Ich implementácia neobsahuje ošetrovanie žiadných špeciálnych prípadov, nakoľko o všetko sa stará nadtrieda. Triedy boli vytvorené, aby reflektovali hierarchiu dedenia jednotlivých Boxov. Nie je však vylúčené, že v budúcne pribudne do tried efektívny kód.

4.5 Demo aplikácia

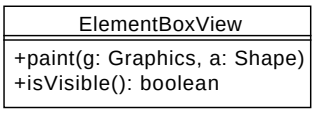
Užívateľské rozhranie aplikácie je rozdelené na dve časti. V hornej sa nachádza textové pole pre zadanie umiestnenia súboru s obsahom na spracovanie (treba zadať úplnú adresu aj s protokolom, napr. `http://www.vutbr.cz`) a tlačítko pre spusteni akcie. V dolnej časti je pomocou prvku JTabbedPane zobrazený výsledok spracovania pomocou troch rôznych *„nástrojov“*. Použitý bol `SwingBox`, `CSSBox` a `JEditorPane + HTMLEditorKit`.

[43] http://download.oracle.com/javase/6/docs/api/java/awt/image/ImageObserver.html

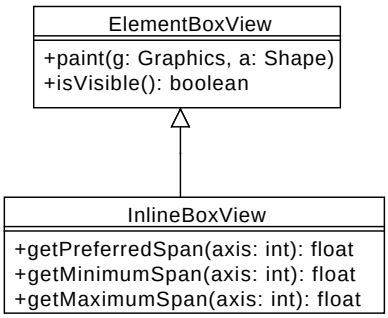
[44] Vysvetlenie vzťahu medzi animovaným gifom a rozhraním ImageObserver: http://www.permadi.com/tutorial/javaImgObserverAndAnimGif/



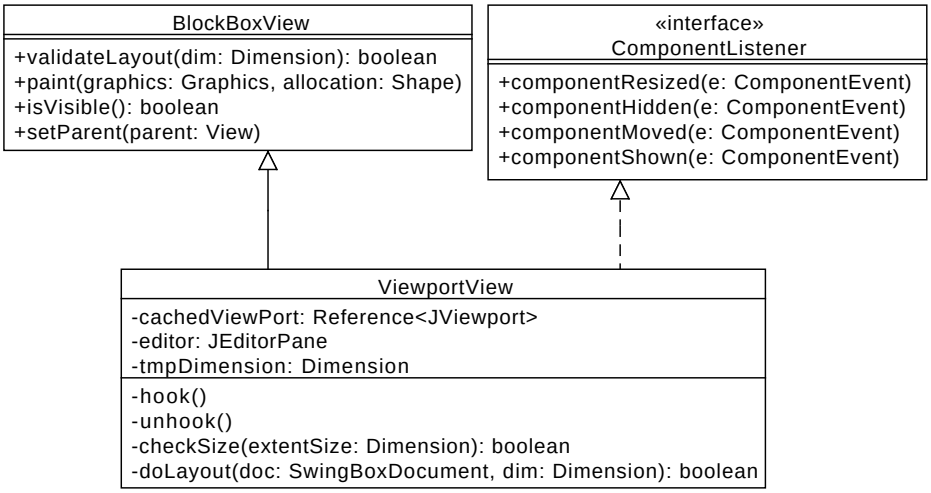
Obrázek 4.16: Diagram triedy ElementBoxView.



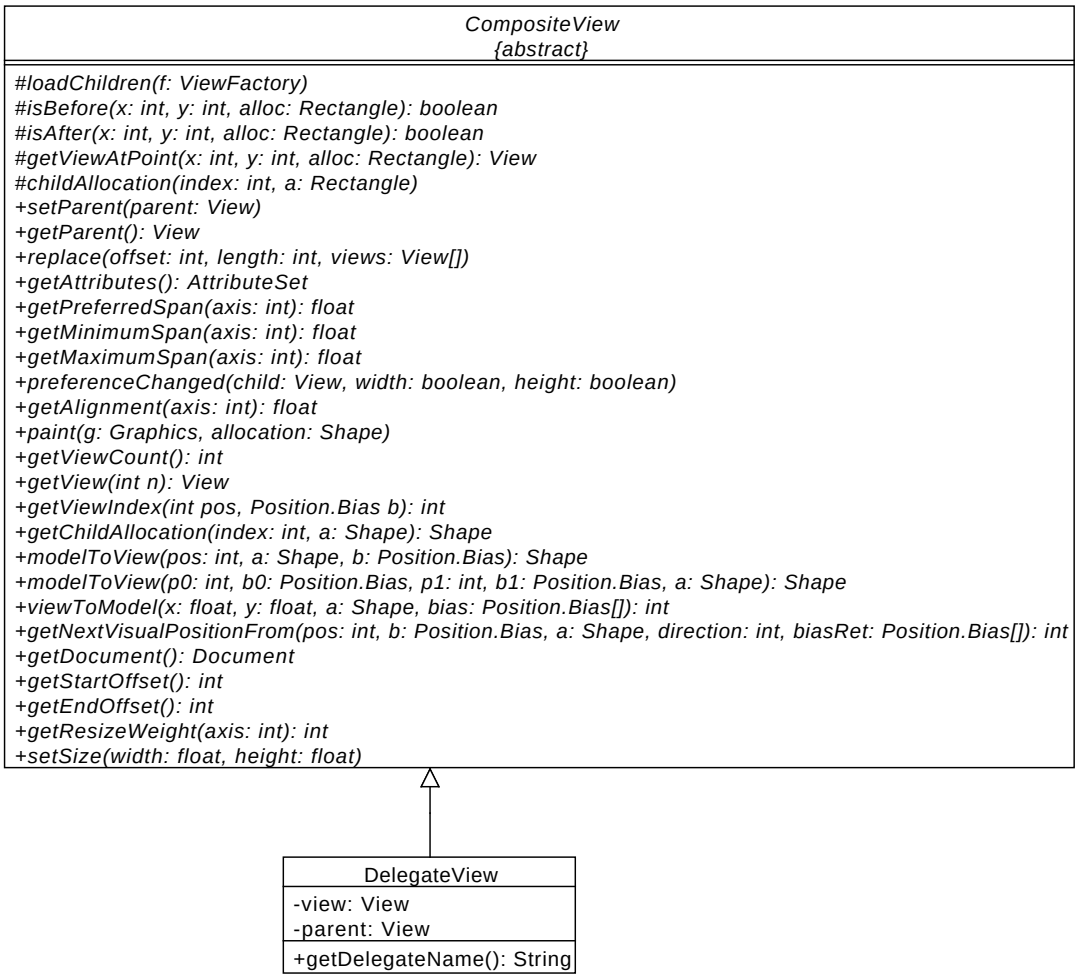
Obrázek 4.17: Diagram triedy BlockBoxView.



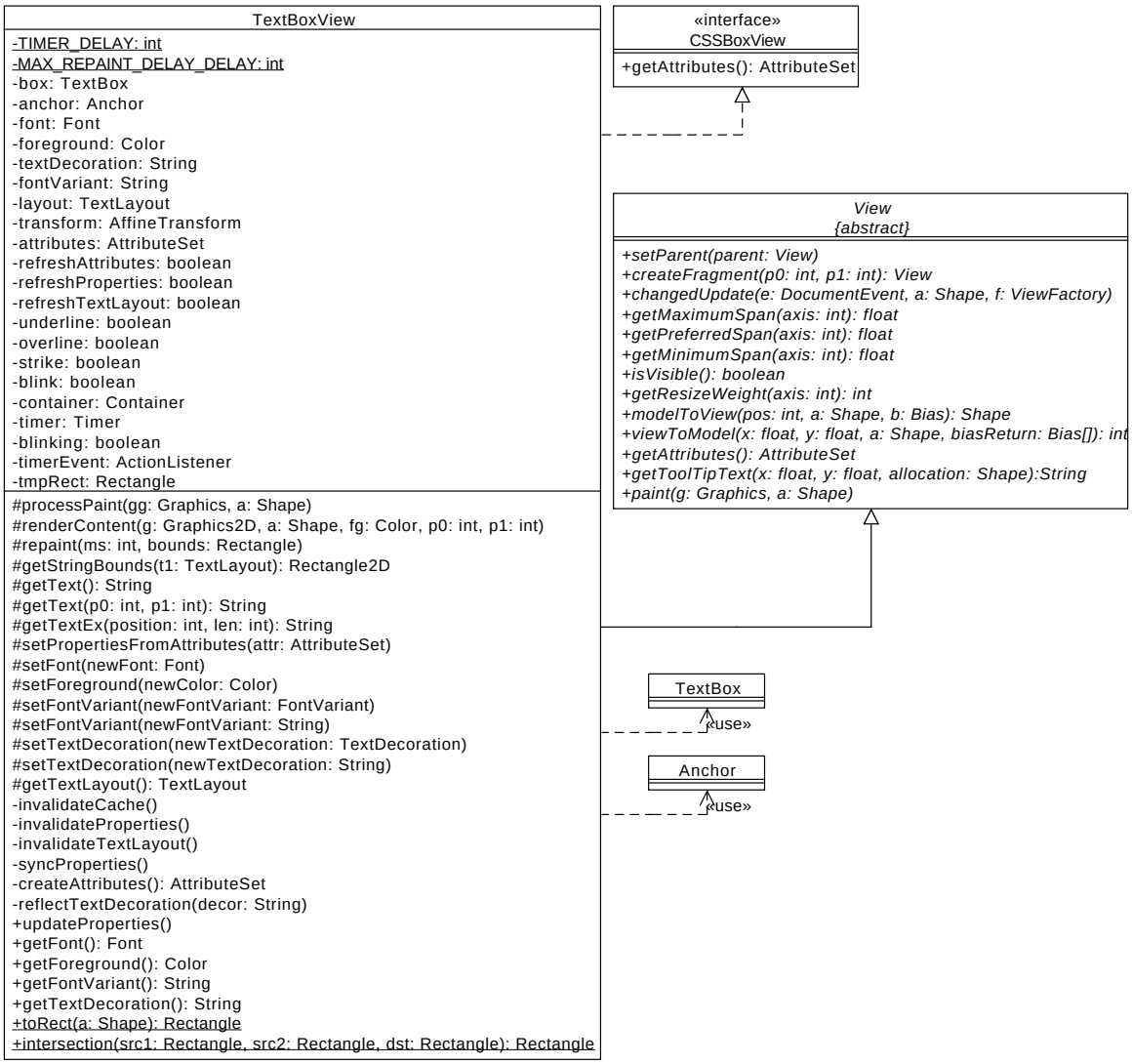
Obrázek 4.18: Diagram triedy InlineBoxView.



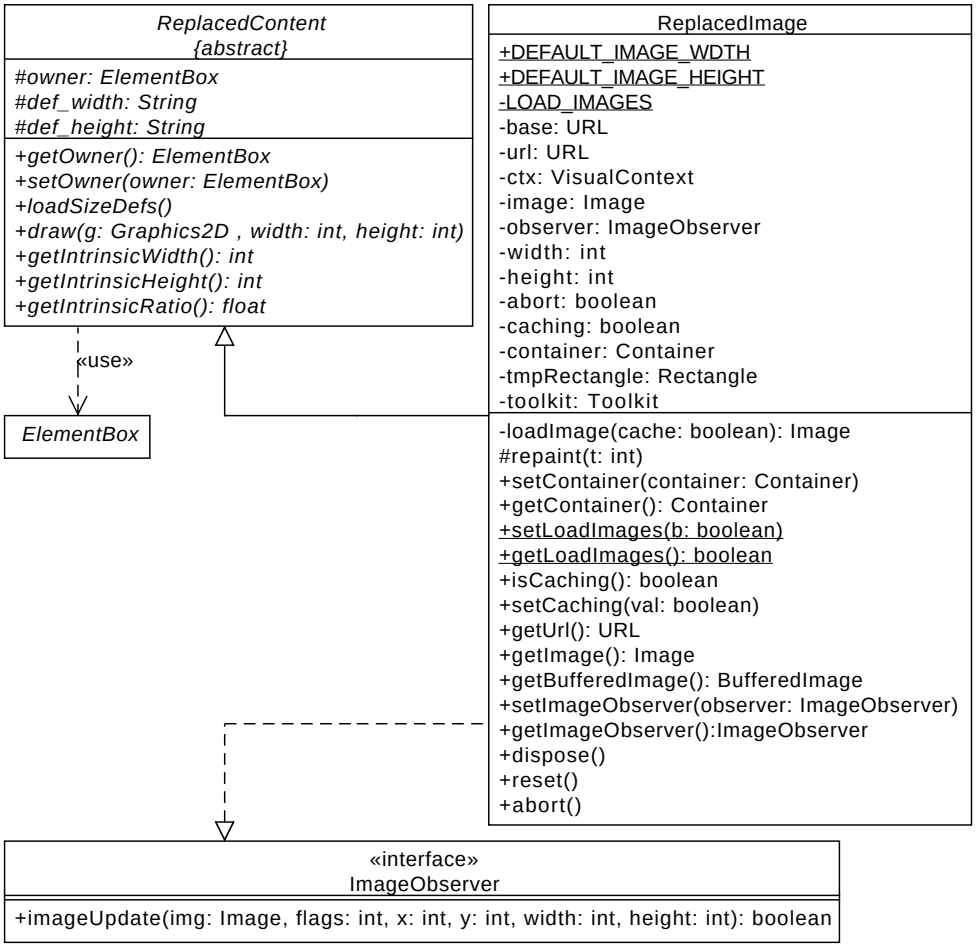
Obrázek 4.19: Diagram triedy ViewportView.



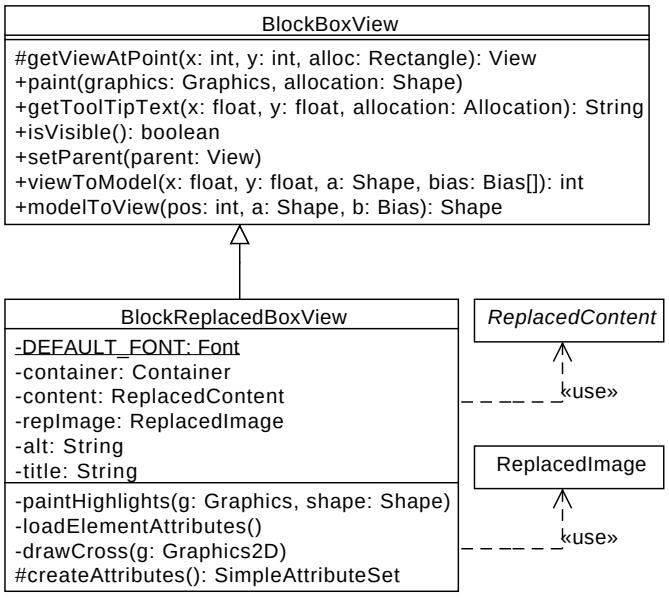
Obrázek 4.20: Diagram triedy DelegateView.



Obrázek 4.21: Diagram triedy TextBoxView.



Obrázek 4.22: Diagram triedy ReplacedContent a ReplacedImage.

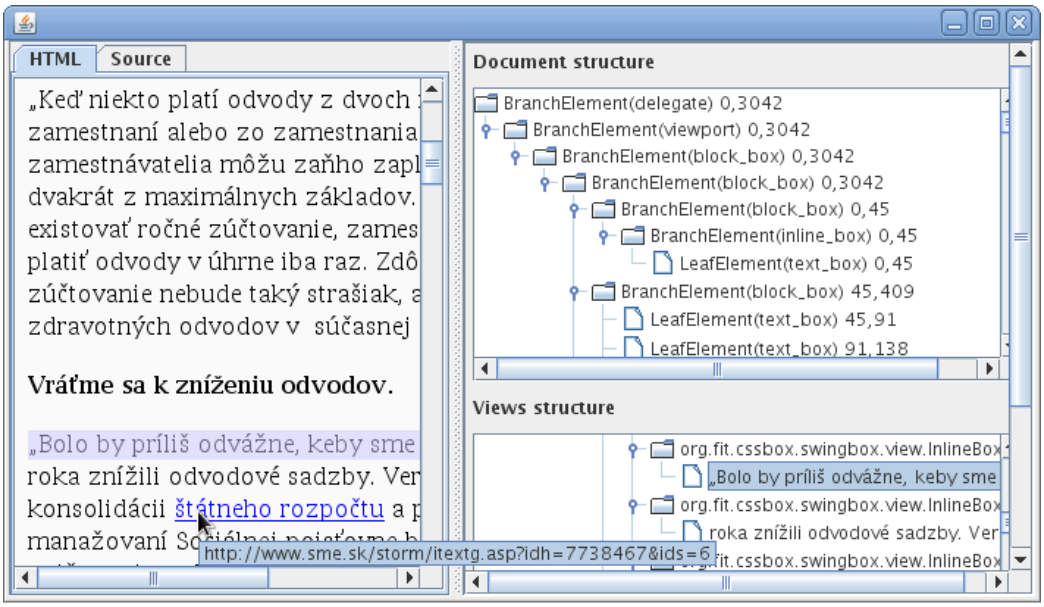


Obrázek 4.23: Diagram triedy BlockReplacedBoxView.

Kapitola 5

Testovanie

Testovanie aplikácie prebiehalo súbežne počas jej vývoja. Neboli použité žiadne špeciálne frameworky či aplikácie ako je *JUnit*, ale len obyčajná metóda `main`, ktorá bola aplikovaná všade tam, kde to bolo potrebné. Bola vytvorená aj jednoduchá aplikácia, ktorá v lavej časti zobrazovala BrowserPane s výsledkom spracovania a na strane druhej, zobrazovala výsledný strom Element objektov a View objektov. Počas vývoja, bolo nazbieraných a vytvorených niekolko testovacích HTML súborov, na ktorých sa vizuálnym spôsobom testovala správnosť vykresľovania obsahu v porovnaní so skutočným internetovým prehliadačom (bola použitá Opera). Tieto súbory je možné nájsť na priloženom CD v zložke test.



Obrázek 5.1: Ukážka testovacej aplikácie s výsledkom v BrowserPane.

5.1 Známe chyby a nedostatky

Pri prijímaní dát zo servera, sú bajty konvertované na znaky. To sa deje podľa uvedenej znakovej sady v http hlavičke (napr. `Content-Type: text/html; charset=iso-2022-jp`). Ak server túto informáciu neposkytne, je použité kódovanie UTF-8, čo sa môže prejaviť ako

„štvorčeky“ či iné artefakty vo výslednom texte. Tomuto by sa dalo zabrániť nastavením pola `Accept-Charset` na hodnotu utf-8 v http hlavičke pri odosielaní požiadavky a dúfať, že server pošle dáta v správnom tvare, alebo zmeniť implementáciu vo vnútri JEditorPanu, aby nevykonával žiadnu konverziu a vstupný prúd bajtov priviesť až k parseru. Niektoré parsery sú schopné detekovať použitú znakovú sadu.
Problémom je aj „vysvietenie textu“, ktoré nepracuje dobre: text zvykne zmiznúť, alebo sa vysvietia iné časti textu. Tento problém zatial nemá riešenie. V prípade *niektorých* animovaných obrázkov typu GIF s transparentným pozadím sa stáva, že predošlý snímok nie je zmazaný a nový je cez neho vykreslený, čo nevyzerá pekne. Snahy o prekreslenie obrázka pozadím neboli úspešné. Treba dodať, že časovanie, ako aj vykresľovanie jednotlivých snímkov má na starosti Java.
Ako bolo spomenuté v bode 4.4.6, ViewportView má obmedzené schopnosti pri zmene rozloženia obsahu. Navyše, v istých prípadoch sa môže stať, že vykreslený obsah nepokrýva celú dostupnú plochu, alebo je useknutý (čo je zapríčinené orezávaním plochy). Dekorácia textu, teda underline, overline, line-through a blink, je obmedzená na použitie len jednej z vymenovaných, pretože CSSBox vo výslednom spracovaní dokáže reprezentovať iba jednu.

Kapitola 6

Záver

CSSBox je vynikajúci v analýze a spracovaní obsahu, no vykresľovacie schopnosti su obmedzené na vytvorenie *„obrázkovej bitmapy“*. Zlepšenie v tejto oblasti by mu prinieslo lepšie zviditeľnenie sa a v konečnom dôsledku aj lepší užívateľský komfort.
Návrh postavený na prvku JEditorPane s úpravami, spolu s vlastnou implementáciou triedy EditorKit, predstavuje najvhodnejšie riešenie. Práca prináša:

- Štandardný, textovo orientovaný komponent.
- Podporu pre odkazy a generovanie HyperlinkEvent udalostí.
- Podporu presmerovania a HTTPS (podmienkou je platný a podpísaný SSL certifikát).
- Generovanie všeobecných udalostí, informačného charakteru – GeneralEvent.
- Zobrazovanie popiskov obrázkov a odkazov.
- Lepšie spracovanie obrázkov pre CSSBox a podporu pre animácie vo formáte GIF (trieda ReplacedImage).
- Možnosť získania zobrazeného obsahu v textovej podobe.
- Miesta pre úpravu funkcionality podľa vlastných predstáv.
- Riešenie s využitím čistej Javy.

V ďalšom vývoji CSSBoxu a SwingBoxu by sa mali odstrániť popísané nedostatky. Zlepšiť by sa mohla rýchlosť spracovania a znížiť pamäťová náročnosť. Napríklad, každý element dostane vlastnú inštanciu triedy Anchor, pričom mnohé elementy nie sú odkazy a tie, ktoré nimi skutočné sú, si kopírujú dáta. Mal by byť vytvorený špeciálny objekt, *„null_anchor“* predstavujúci prázdny odkaz. Elementy, ktoré nie sú odkazom, by zdieľali referenciu na tento objekt a ostatné elementy by zdieľali referenciu na Anchor s dátami navzájom. Ďalším zlepšením by bola implementácia ovládacích prvkov ako sú tlačítka či textové polia. S výhľad do ďalekej budúcnosti by mohla pribudnúť podpora pre Javascript.

Literatura

[1] Bos, B.; Çelik, T.; Hickson, I.; aj.: Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification [online]. W3C Recommendation, http://www.w3.org/TR/2011/PR-CSS2-20110412, 2011.
URL http://www.w3.org/TR/2011/PR-CSS2-20110412

[2] Burget, R.: CSSBox Manual [online]. http://cssbox.sourceforge.net/manual/, [cit. 2010-12-27].

[3] Fowler, A.: Mixing heavy and light components [online]. http://java.sun.com/products/jfc/tsc/articles/mixing/index.html, 2003-04-12 [cit. 2010-12-19].

[4] Fowler, A.: A Swing Architecture Overview [online]. http://java.sun.com/products/jfc/tsc/articles/architecture/, 2003-04-12 [cit. 2010-12-21].

[5] Isenhour, P.: Faster JTextPane Text Insertion (Part II) [online]. http://javatechniques.com/blog/faster-jtextpane-text-insertion-part-ii/, [cit. 2011-04-29].

[6] Loy, M.; Eckstein, R.: *Java Swing:* . Java Series, O'Reilly, 2002, ISBN 9780596004088.
URL http://books.google.com/books?id=W3HjIAduQfkC

[7] Prinzing, T.: Using the Swing Text Package [online]. http://java.sun.com/products/jfc/tsc/articles/text/overview/, 2003-04-12 [cit. 2011-04-02].

[8] Richardson, W.; Avondolio, D.; Schrager, S.; aj.: *Professional Java JDK 6 Edition.* Programmer to programmer, Wiley Technology Pub., 2006, ISBN 9780471777106.
URL http://books.google.com/books?id=eWQPYg6wldsC

[9] WWW stránky: The JComponent Class [online]. http://download.oracle.com/javase/tutorial/uiswing/components/jcomponent.html, [cit. 2010-12-19].

[10] WWW stránky: Text Component Features [online]. http://download.oracle.com/javase/tutorial/uiswing/components/generaltext.html, [cit. 2010-12-22].

Dodatek A

Obsah CD

Na priloženom CD sa nachádza:

- readme.txt - komentár obsahu CD.
- src - zdrojové kódy pre SwingBox spolu s kódom pre CSSBox. Kód CSSBoxu môže obsahovať drobné úpravy.
- bin - obsahuje výsledné binárne súbory.
- cssbox - obsahuje originálne, nezmenené zdrojové i binárne súbory pre CSSBoxu.
- javadoc - vývojárska dokumentácia SwingBoxu vo forme JavaDoc (možné vygenerovať zo zdrojových kódov).
- test - zbierka HTML kódov, použitých pri testovaní.
- thesis - obsahuje túto diplomovú prácu v elektronickej forme.
- backup - komprimované zdrojové kódy.

Dodatek B

Ukážka použitia

V tejto kapitole bude prebraná jednoduchá ukážka použitia SwingBoxu.
Predpokladáme, že kód sa vykonáva v rámci metódy. Celé spracovanie je zastrešené komponentom BrowserPane, ktorý vytvoríme.

```
BrowserPane browser = new BrowserPane();
```

Po tomto volaní má browser registrovaný SwingBoxEditorKit, spracovanie obsahu nastavené na text/html a aktivované zobrazenie tooltip popiskov.
Ďalej budeme požadovať paralelné spracovanie dokumentu, podľa možností s najvyššou prioritou.

```
System.setProperty(Constants.DOCUMENT_ASYNCHRONOUS_LOAD_PRIORITY_PROPERTY,
    Integer.toString(Thread.MAX_PRIORITY));
```

Po rokoch sa API CSSBoxu zmenilo a DefaultAnalyzer by nepracoval správne. Potrebujeme SwingBoxu odovzdať našu vlastnú implementáciu rozhrania CSSBoxAnalyzer. To je možné priamim zadaním:

```
browser.setCSSBoxAnalyzer(new FastAnalyzer());
```

Prípadne cez vlastnosti prostredia:

```
System.setProperty(Constants.DEFAULT_ANALYZER_PROPERTY,
    ''org.company.project.FastAnalyzer'');
```

K užívateľskému komfortu patria funkčné odkazy. Niekedy je vhodné získať potvrdenie od užívateľa či si skutočne želá prejsť na dané umiestnenie:

```
HyperlinkListener handler = new DefaultHyperlinkHandler(){
    protected void loadPage(JEditorPane pane, HyperlinkEvent evt) {
        if (getPermission(evt)) {
            super.loadPage(pane, evt);
        }
    }
};

browser.addHyperlinkListener(handler);
```

SwingBox generuje GeneralEvent udalosti, ktoré môžu obsahovať zaujimavé informácie. Napríklad môžme zistiť čas spracovania stránky:

```
class GeneralEventHandler implements GeneralEventListener {
    private long time;

    public void generalEventUpdate (GeneralEvent e){
        if (e.event_type == EventType.page_loading_begin) {
            time = System.currentTimeMillis();
        } else if (e.event_type == EventType.page_loading_end) {
            System.out.println(''Loaded in ''
                + (System.currentTimeMillis() - time) + '' ms'');
        }
    }
}

browser.addGeneralEventListener(new GeneralEventHandler());
```

Zadanie stránky na spracovanie je jednoduché. Stačí použiť metódu setPage ak vieme URL, alebo setText, ak máme len HTML kód stránky ako reťazec.

```
browser.setPage(new URL(''http://www.phoronix.com''));
```